RC-005

MANET プロトコルのモデル検査のための状態空間削減手法
A State Space Reduction Method for Model Checking of MANET Protocols

長島 悠太 * **小島 英春** * **土屋 達弘** *
Yuta Nagashima Hideharu Kojima Tatsuhiro Tsuchiya

1. はじめに

Mobile Ad hoc Network(MANET) では，ネットワークに存在する端末が周囲の端末と通信を行い，自律的にネットワークを構築する．端末は移動することが可能であり，ネットワークからの離脱，参加が生じることを許可している．MANET では，端末の移動により，隣接端末の変化や通信経路変更が生じるため，MANET プロトコルでは，通信経路を維持することが必要である．多くのプロトコルが提案されており [1][2]，それらは，AODV[3] や DSR[4] に代表される，送信元と宛先間に通信要求が発生した場合に経路を構築するリアクティブ型のプロトコルと，OLSR[5] や TBRPF[6] に代表される端末間の経路を常に維持しているプロアクティブ型プロトコルの 2 種類に分類することができる．

近年，自動車の自動運転の研究が行われており，実車による実験も行われている．MANET の 1 つとして，対象とする端末を自動車に注目した VANET(Vhiecular Ad hoc Networks) ルーティングプロトコルは，多く研究されており [7][8]，隣接端末の動的な変化やネットワークへの端末の加入，離脱が頻繁に行われるこのようなネットワークに適している．このようなプロトコルが社会で用いられる場合，不具合の発生は，社会的な問題に発展する可能性がある．

そこで，プロトコルが正しく動作するかどうか，不具合が存在しないかどうかを確かめる方法の 1 つとして，**モデル検査**が挙げられる．モデル検査は，対象とするプロトコルが，想定される全ての状態において与えられた特性を満たすかどうか，また，それらの状態において不具合が生じないかどうかを検査する．本論文ではリアクティブ型のプロトコルの 1 つである AODV のモデル検査を試みる．AODV の経路構築の振舞をモデル化し，モデル検査器 SPIN[9] を用いてモデル検査を行う．MANET プロトコルでは，端末の移動により多数のトポロジが発生する．全ての可能なトポロジを検査することは，実機やシミュレーションでは，困難であるため，全てのトポロジを網羅するモデル検査は，非常に有用である．しかし，モデル検査では，しばしば状態爆発の問題が生じる．特に MANET プロトコルでは，端末数が多くなると，それに伴いトポロジが莫大に増加する．そのため，探索すべき状態数も非常に増加し，実行時間の増加やメモリ利用量の増加を引き起こす．その結果，現実的な時間でモデル検査が終了しないことや，全ての状態を探索できない場合がある．

送信元は，経路要求パケットをブロードキャストした後に，返信パケットを受信するまで，一定期間待機する．この待機期間を経過すると，宛先までパケットが届かなかったとみなし，改めて，経路要求パケットを送信する．

*大阪大学大学院情報科学研究科, Graduate School of Information Science and Technology, Osaka University

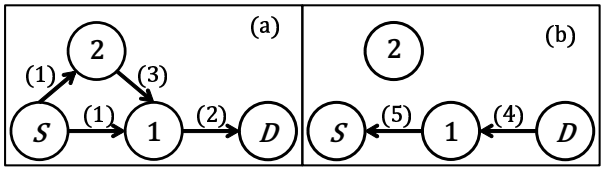

図 1: 経路構築の 1 例

AODV では，この試みを 3 回行う．3 回の経路構築をモデル検査することは，状態数が増加する要因となる．[10] では，3 回の経路構築の試みをそれぞれ，1 ラウンド，2 ラウンド，3 ラウンドとし，各ラウンドにおいて，各端末が取りうる経路表のエントリに注目し，3 ラウンドのみを行うことで，モデル検査を行うことが可能であることを示している．しかし，[10] では，どのようなネットワークを対象にしているかが明確にしておらず，端末の振舞も限定的である．本論文では，対象とするネットワークを明確にし，端末がクラッシュと復帰を行う場合を考慮したモデル検査を行う．[10] では，状態空間の全探索に用いた Promela コードにループ検出のための C 言語の埋め込みに加えて Promela コード自体にも改変を加えることで，ループの検出を行っていたが，本論文では，状態空間の全探索に用いた Promela コードに C 言語を埋め込むだけでループの検出を行うことが可能である．

2.AODV

本研究で対象とする AODV について説明を行う．AODV は，オンデマンド型の MANET ルーティングプロトコルである．送信元端末は，通信要求が発生すると，その宛先端末への経路構築を開始する．送信元宛先間の経路が構築され，通信が完了し，その経路が利用されなくなると，その経路情報は破棄される．各端末は経路表を保有しており，その経路表は，宛先，宛先への次ホップ，宛先からのホップ数，シーケンス番号の情報を表すエントリが保存されている．このエントリは，経路構築にあたって用いられる，経路要求パケット *rreq*，*rreq* への返信として送信される *rrep* の情報を用いて生成される．

AODV の経路構築例

ここでは，図 1 に示すような，端末数が 4 のネットワークを用いて経路構築の例を挙げる．ネットワークには，送信元端末 S と宛先端末 D が存在し，端末 1 と端末 2 は中継端末である．AODV は，送信元が経路構築を行うにあたって，経路要求パケット *rreq* を送信する．それを受信した端末は，そのパケットの宛先が自身でない場合，それを転送する．*rreq* を受信した宛先は，その返信として，*rrep* パケットをユニキャストで送信する．*rrep* は，*rreq* が転送されて来た経路を送信元に向けてユニキャストで送信される．図 1(a) は，*rreq* がブロードキャストされ，宛先まで届くまでを表している．図 1(b) は，*rrep* が返信され，送信元まで届くまでを表している．

経路構築のプロセスは以下の通りである．この例において，各端末が保有する経路表 rt_S,rt_D,rt_1,rt_2 は空である．$rreq$ のシーケンス番号 seq_S と seq_D はどちらも 0 とする．

(a)-(1): 送信元 S が $rreq$ をブロードキャストし，端末 1，端末 2 がそれを受信する．端末 1，端末 2 は，受信した $rreq$ を用いて，エントリを生成し経路表に保存する．

(a)-(2): 端末 1 は，$rreq$ の宛先ではないため，そのパケットをブロードキャストする．

(a)-(3): 端末 2 は，$rreq$ の宛先ではないため，そのパケットをブロードキャストする．

(b)-(4): 宛先 D は，$rreq$ を受信した後，経路表へエントリを追加し，送信元 S への返信である $rrep$ をユニキャストで送信する．

(b)-(5): $rrep$ を受信した端末 1 は，経路表に D 宛のエントリを保存し，経路表から S 宛のエントリを用いて，S へパケットをユニキャストで送信する．

送信元 S は，端末 1 からの $rrep$ を受信することで，経路表 rt_S に D 宛のエントリを保有することとなる．

3. モデル

ここでは，本論文で対象とするネットワークや端末のモデルについて説明し，それらをどのように Promela を用いて SPIN 上で実行可能なモデルにするかを説明する．

3.1. ネットワークと端末

本論文で対象とするネットワークと端末について述べる．このネットワークは，$k(k > 2)$ 個の端末から構成されるネットワークであり，1 つの送信元端末と 1 つの宛先端末，$k-2$ 個の中継端末から構成される．全ての端末はパケットを受信するためのチャネルを有し，中継端末と宛先端末は，経路情報を保存する経路表を有している．端末がパケットを送信すると，そのパケットは他の端末に受信されるまで内容は変化しない．端末はクラッシュする可能性があり，クラッシュした後に復帰する．端末は，多くても 1 回しかクラッシュしないとする．クラッシュから復帰した端末は，チャネルに保有していたパケット，経路表の情報が初期化され空になる．端末はブロードキャストとユニキャストによってパケットの送信を行う．ブロードキャストされたパケットは，通信範囲内に存在する端末に受信される．ユニキャストによって送信されたパケットは，ユニキャストの対象となる端末が通信範囲内に存在する場合に受信される．MANET では，送信時に通信範囲内に存在したとしても，パケットの衝突や端末の移動が発生するため，端末がそのパケットを受信することを保証しない．この様なネットワーク上で動作する AODV を SPIN で検証するにあたって，Promela を用いてモデル化を行う．

3.2.SPIN モデル

本節では，AODV を SPIN で検証するにあたって，送信元端末や中継端末，宛先端末を表すプロセスとチャネルや経路表をどのように Promela で定義しているか，ブロードキャストや端末の移動をどのように Promela を用いて表現しているかについて説明する．使用される定数 SRC_NODES は送信元端末数を，NODES は中継端末と宛先端末を合わせた数を表している．

```
 1 :proctype Src(byte i,d){
 2 : byte id=i,index=i,cnt=0;
 3 : byte sent[SRC_NODES+NODES];
 4 : bool found=false;
 5 : packet pkt,tmp;
 6 : pkt.seq=0;
 7 : do::(!found)->
 8 :   pkt.type=0;pkt.sndr=id;
 9 :   pkt.src=id;pkt.dest=d;pkt.ttl=10;
10 :   if::(pkt.seq<3)->pkt.seq++;
11 :     broadcast();
12 :     ::else->break;
13 :   fi;
14 :   do::c[id-1]?pkt->
15 :     atomic{
16 :     if::((pkt.type==1)&&(pkt.next==id)
17 :       &&(pkt.dest==id))->found=1;break;
18 :       ::else->skip;
19 :     fi;
20 :     }
21 :     ::timeout->skip;
22 :   od;
23 : od;
24 :}
```

図 2: 送信元端末を表わす Promela コード

3.2.1. チャネル

端末を表すプロセス p_{id} は，以下の様に定義されたチャネル c[id] を保有する．定数 BUFF はチャネルのバッファを表し，このモデルでは 1 である．packet の type は 0 であれば $rreq$，1 であれば $rrep$ を表す．

```
typedef packet{byte type,sndr,next,src,
        dest,ttl;int seq};
chan    c[SRC_NODES+NODES] = [BUFF] of {packet};
```

3.2.2. 経路表

中継端末と宛先端末のプロセス p_{id} が保有する経路表は，以下に定義された tables[id] を経路表として持つ．定数 RT_LENGTH は経路表が保有するエントリの最大数を表している．

```
typedef entry{byte seq,dest,next};
typedef table{entry entries[RT_LENGTH]};
table   tables[NODES];
```

3.2.3. 送信元端末を表すプロセス

送信端末を表すプロセスを図 2 に示す．図 2 は 7 行目から 23 行目までを繰り返す．8，9 行目で $rreq$ を生成し，10 行目でシーケンス番号を 1 増加する．その後にブロードキャストを行う．ブロードキャスト後は，パケットの受信を行い，受信したパケットが $rrep$ かつ自身宛であるならば経路が構築できたとみなす．

3.2.4. 中継端末，宛先端末のモデル

送信元端末以外の端末は，中継端末か宛先端末となる．そのプロセスを表す Promela コードを図 3 に示す．6 行目から 28 行目までを繰り返す．受信後は，クラッシュの判定とパケットの処理を行っている．12 行目から 17 行目では，受信したパケットの情報を用いて経路表の操作を行っており，パケットを転送する必要がある場合は，送信するパケットの更新を行い，forward を 1 にする．19 行目から 27 行目は転送すべきパケットが存在する場合，パケットの type によって，ブロードキャストで送信するかユニキャストで送信するかを決定する．

```
 1 :proctype Node(byte i){
 2 : byte id=i,index=i,cnt=0,crash=0;
 3 : byte sent[SRC_NODES+NODES];
 4 : packet pkt,tmp;
 5 : bool exist_table=0,forward=0;
 6 : do::c[id-1]?pkt->
 7 :  atomic{
 8 :   if::(crash==0)->
 9 :     //クラッシュするかどうかの判定と,
10 :     //クラッシュした場合の復帰処理
11 :   fi;
12 :   if::(pkt.type==0)->
13 :       //rreq を受信した場合の処理
14 :       //宛先が自身の場合 rrep の生成を行う
15 :     ::(pkt.type==1)->
16 :       //rrep を受信した場合の処理
17 :   fi;//経路表 tables[id] の操作も行う
18 :  }
19 :  if::(forward==1)->//転送処理
20 :    //rreq の場合はブロードキャスト
21 :    //rrep の場合はユニキャスト
22 :    if::(pkt.type==0)->broadcast();
23 :      ::(pkt.type==1)->unicast(pkt.next);
24 :      ::else->skip;
25 :    fi;
26 :    ::else->skip;
27 :  fi;
28 : od;
29 :}
```

図 3: 中継・宛先端末を表す Promela コード (抜粋)

```
1 :inline unicast(uni){
2 : if::(len(c[uni])==CHAN_BUF)->
3 :   c[uni]?tmp;
4 :   ::else->skip;
5 : fi;
6 : c[uni]!pkt;
7 :}
```

図 4: ユニキャストの処理を行う Promela コード

3.2.5. ブロードキャストとユニキャスト

ユニキャスト

ユニキャストは，図 4 に示すコードにより実行される．uni は，pkt を受信する端末の *id* を表しているため，その端末に対応したチャネルにパケットを保存している．チャネルに保存できない場合は，保存されているパケットと入れ替える．

ブロードキャスト

図 5 は，パケットのブロードキャストを行う Promela コードである．MANET では，ブロードキャストされたパケットを受信する端末は，パケットを送信した端末の通信範囲内に存在する端末である．図 5 は端末数が 5 つの場合に，端末の移動とブロードキャストをモデル化した Promela コードである．図 5 の *AROUND* は，最大でいくつの端末が通信範囲内に存在するかを表している．変数 sent[] は，送信済みの端末を表す．3 行目から 6 行目において sent[] の内容は，0 に初期化される．7 行目は，送信端末自身を送信対象からはずすための処理である．端末の移動により，パケットを送信する端末は変化するため，10 行目から 16 行目の様に，送信先を非決定的に選択する．送信先となる端末が決定すると，19 行目で送信済みとし，20 行目でユニキャストで送信対象の端末へパケットを送信する．もし，10 から 16 行目において，index=255 が選択された場合は，そのループで

```
 1 :inline broadcast(){
 2 : atomic{
 3 :  do::(cnt<(SRC_NODES+NODES))->
 4 :      sent[cnt]=0;cnt++;
 5 :    ::(cnt==(SRC_NODES+NODES))->break;
 6 :  od;
 7 :  sent[id]=1;
 8 :  cnt=0;
 9 :  do::(cnt< AROUND)->
10 :    if::(sent[0]==0)->index=0;
11 :      ::(sent[1]==0)->index=1;
12 :      ::(sent[2]==0)->index=2;
13 :      ::(sent[3]==0)->index=3;
14 :      ::(sent[4]==0)->index=4;
15 :      ::index=255;
16 :    fi;
17 :    cnt++;
18 :    if::(index==255)->skip;
19 :      ::else->sent[index]=1;
20 :        unicast(index);
21 :    fi;
22 :    ::else->break;
23 :  od;
24 : }
25 :}
```

図 5: ブロードキャストを表す Promela コード

は，パケットが送信されない．このようにユニキャストを複数回実行することによりブロードキャストを実現している．

4. 提案法

AODV は，経路を構築する際に，*rreq* パケットをブロードキャストする．ある一定期間内に，*rrep* を宛先端末から受信しない場合，改めて，*rreq* パケットをブロードキャストする．AODV では，3 回経路構築を試みることで，宛先までの経路構築を行う．

モデル検査において，1 回目から 3 回目までの全ての振舞を 1 回のモデル検査で行うと，状態数が非常に大きくなり，現実的な時間で検査が終了しないことがある．また，状態数が非常に大きいため，必要なメモリが尽きてしまうこともある．そこで，本研究では，1 回の経路構築だけをモデル検査することで，3 回の経路構築全体を検証する方法を提案する．1 回の経路構築を 1 ラウンドと表現する．

ラウンドの定義: 1 ラウンドは，送信元が *rreq* パケットをブロードキャストすることで開始する．第 n ラウンドに送信元が送信する *rreq* を $rreq_S^n$ とする．

図 6 の S^1 は，第 1 ラウンド開始の状態の集合を表し

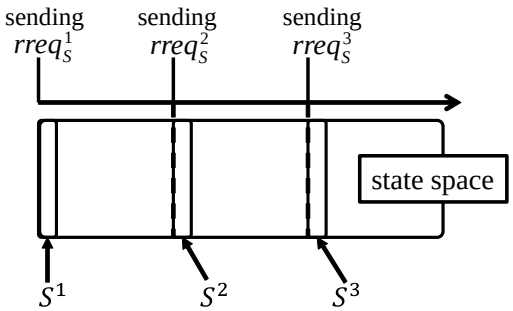

図 6: 状態空間

表 1: パケットから生成されるエントリ

received packet	generated entry
$rreq_i^n$ $(S, i, D, seq_S, seq_D, hops)$	$e^n_{(req,i)}$ $(S, i, seq_S, hops)$
$rrep_j^n$ $(D, j, S, seq_D, hops)$	$e^n_{(rep,j)}$ $(D, j, seq_D, hops)$

ている．S^2，S^3 はそれぞれ第 2 ラウンド，第 3 ラウンド開始の状態を表している．S^1 の各端末の経路表の取り得るエントリと S^2 の各端末の経路表の取り得るエントリが S^3 の各端末の経路表の取り得るエントリに含まれることを示すことで，第 3 ラウンドのみの経路構築のモデル検査で検証可能であることを示す．

提案法では，以下の仮定を用いる．

・経路表のエントリの削除は行わない．
・SD ペアは 1 組
・中継端末がリプライパケットの返信を行わない．

提案法の手順は次の通りである．

STEP 1: 各ラウンドで存在するパケットと経路表の取り得るエントリを列挙する．

STEP 2: プロセスが受信するパケットと経路表の関係を示す．

STEP 3: 3 ラウンド開始時の経路表の組み合わせを作成する．

STEP 4: それぞれの組み合わせに対して，モデル検査を行う．

4.1. 経路表に含まれるエントリ

各ラウンドにおいて，経路表がどのようなエントリを保持するかについて述べる．表 1 に受信したパケットから生成されるエントリを示す．n はラウンドを表し $n \in \{1,2,3\}$ である．i，j は送信した端末を表し，$i \in \{1,2,\ldots,S\}$，$j \in \{1,2,\ldots,D\}$，$i \neq j$ である．3 ラウンドが始まる前に取りうる経路表は，以下の 6 通りである．

$rt1$: $\emptyset$　　$rt2$: $\{e^1_{(req,i)}\}$　　$rt3$: $\{e^1_{(req,i)}, e^1_{(rep,j)}\}$
$rt4$: $\{e^2_{(req,i)}\}$ $rt5$: $\{e^2_{(req,i)}, e^1_{(rep,j)}\}$ $rt6$: $\{e^2_{(req,i)}, e^2_{(rep,j)}\}$

$rt1$ から $rt6$ の経路表が取り得るエントリの組合せは，どのラウンドでどのパケットを受信したかによって変化する．それぞれの経路表が取り得るエントリについて説明する．

$rt1 : \emptyset$
この経路表 $rt1$ のエントリは，3 ラウンド開始までの間に，1 度もパケットを受信しなかったことを表す．

$rt2 : \{e^1_{(req,i)}\}$
この経路表のエントリは，3 ラウンド開始時までに 1 回は $rreq_i^1$ のパケットを受信し，そのエントリを保有していることを表している．

$rt3 : \{e^1_{(req,i)}, e^1_{(rep,j)}\}$
この経路表のエントリは，3 ラウンド開始時までに，$rreq_i^1$ のパケットを 1 回，$rrep_j^1$ のパケットを 1 回受信している．

$rt4 : \{e^2_{(req,i)}\}$
この経路表のエントリは，3 ラウンド開始時までに 1 回は $rreq_i^2$ のパケットを受信し，そのエントリを保有していることを表している．

表 2: パケット受信前と受信後のエントリ

パケット受信前のエントリ	受信パケット	パケット受信後のエントリ
$rt1$: $\emptyset$	$rreq_k^1$	$rt2 : \{e^1_{(req,k)}\}$
	$rrep_l^1$	$rt1 : \emptyset$
	$rreq_k^2$	$rt4 : \{e^2_{(req,k)}\}$
	$rrep_l^2$	$rt1 : \emptyset$
$rt2$: $\{e^1_{(req,i)}\}$	$rreq_k^1$	$rt2 : \{e^1_{(req,i)}\}$
	$rrep_l^1$	$rt3 : \{e^1_{(req,i)}, e^1_{(rep,l)}\}$
	$rreq_k^2$	$rt4 : \{e^2_{(req,k)}\}$
	$rrep_l^2$	$rt2 : \{e^1_{(req,i)}\}$
$rt3$: $\{e^1_{(req,i)}, e^1_{(rep,j)}\}$	$rreq_k^1$	$rt3 : \{e^1_{(req,i)}, e^1_{(rep,j)}\}$
	$rrep_l^1$	$rt3 : \{e^1_{(req,i)}, e^1_{(rep,j)}\}$
	$rreq_k^2$	$rt4 : \{e^2_{(req,k)}, e^1_{(rep,j)}\}$
	$rrep_l^2$	$rt3 : \{e^1_{(req,i)}, e^1_{(rep,j)}\}$
$rt4$: $\{e^2_{(req,i)}\}$	$rreq_k^1$	$rt4 : \{e^2_{(req,i)}\}$
	$rrep_l^1$	$rt5 : \{e^2_{(req,i)}, e^1_{(rep,l)}\}$
	$rreq_k^2$	$rt4 : \{e^2_{(req,i)}\}$
	$rrep_l^2$	$rt6 : \{e^2_{(req,i)}, e^2_{(rep,l)}\}$
$rt5$: $\{e^2_{(req,i)}, e^1_{(rep,j)}\}$	$rreq_k^1$	$rt5 : \{e^2_{(req,i)}, e^1_{(rep,j)}\}$
	$rrep_l^1$	$rt5 : \{e^2_{(req,i)}, e^1_{(rep,j)}\}$
	$rreq_k^2$	$rt5 : \{e^2_{(req,i)}, e^1_{(rep,j)}\}$
	$rrep_l^2$	$rt6 : \{e^2_{(req,i)}, e^2_{(rep,l)}\}$
$rt6$: $\{e^2_{(req,i)}, e^2_{(rep,j)}\}$	$rreq_k^1$	$rt6 : \{e^2_{(req,i)}, e^2_{(rep,j)}\}$
	$rrep_l^1$	$rt6 : \{e^2_{(req,i)}, e^2_{(rep,j)}\}$
	$rreq_k^2$	$rt6 : \{e^2_{(req,i)}, e^2_{(rep,j)}\}$
	$rrep_l^2$	$rt6 : \{e^2_{(req,i)}, e^2_{(rep,j)}\}$

$rt4 : \{e^2_{(req,i)}\}$
この経路表のエントリは，3 ラウンド開始時までに 1 回は $rreq_i^2$ のパケットを受信し，そのエントリを保有していることを表している．

$rt5 : \{e^2_{(req,i)}, e^1_{(rep,j)}\}$
この経路表のエントリは，$rt3$ の経路表になったのちに，1 回は $rreq_i^2$ のパケットを受信したものを表している．

$rt6 : \{e^2_{(req,i)}, e^2_{(rep,j)}\}$
この経路表は $rt4$ または，$rt5$ の後に，$rrep_j^2$ のパケットを受信したことを表している．

端末のクラッシュが各ラウンドにおいて発生した場合の影響について説明する．端末がクラッシュからの復帰時に経路表のエントリが初期化されるため，その端末のエントリは $rt1$ である．1 ラウンドでクラッシュした場合，3 ラウンド開始までに受信する可能性のあるパケットは $rreq_i^1, rrep_j^1, rreq_i^2, rrep_j^2$ の 4 つである．2 ラウンドでクラッシュした場合も，3 ラウンド開始までに受信する可能性のあるパケットは $rreq_i^1, rrep_j^1, rreq_i^2, rrep_j^2$ の 4 つである．それは，2 ラウンドが開始した後も，1 ラウンドで送信されたパケットはネットワークに存在する可能性が存在するためである．よって，3 ラウンド開始時点の経路表のエントリは，端末のクラッシュが発生したとしても $rt1$ から $rt6$ のいずれかとなる．

4.2. ネットワークに存在するパケットの扱い

提案法によって作成されたモデルは 3 ラウンド目を対象にしているため，このモデルのモデル検査開始時には，

各端末のチャネルにパケットは存在しない．しかし，実際の環境を考えると 3 ラウンド開始時にネットワーク内には，1 ラウンドと 2 ラウンドに送信されたパケットが存在する可能性がある．これらのパケットが存在しなくても，3 ラウンド目のモデル検査を行うことができることを説明する．

ネットワークに存在するパケットは $rreq_k^1$，$rreq_k^2$，$rrep_l^1$，$rrep_l^2$ である．k,l はパケットを送信した端末を示し，$k \in \{1,2,\ldots,S\}$,$l \in \{1,2,\ldots,D\}$ である．前述した，取りうる経路表のエントリと，4 つのパケットの関係について述べる．表 2 に取り得る経路表と，$rreq_k^1$，$rreq_k^2$，$rrep_l^1$，$rrep_l^2$ の受信間と受信後の経路表のエントリを示す．前述した経路表 $rt1$ から $rt6$ を保有する端末が，1 ラウンドと 2 ラウンドに送信されたパケットを受信したとしても，$rt1$ から $rt6$ の経路表のエントリとなる．よって，3 ラウンド開始時点でネットワークに存在するパケットを考慮する必要がない．モデル検査を行うにあたって，3 ラウンド開始時に，チャネルを空として扱うことが可能である．

4.3. モデル検査の実行

前述した，各端末の経路表が取り得るエントリの組合せを全てモデル検査を行う．中継端末のプロセス p_k の取り得る経路表のエントリの値の集合を，RT_k とすると，経路表のエントリの組合せは，$RT_1 \times RT_2 \times \ldots RT_N$ である．送信元は，$rreq$ を送信した後は，$rrep$ を受信するまで，経路構築に関与しない．また，宛先端末は，SD ペアが 1 つの場合は，送信元へのエントリのみを保有するため，その内容は，$rt1$，$rt2$，$rt4$ のいずれかである．よって，ネットワーク存在する端末数を N とすると，各端末の経路表が取り得るエントリの組合せは $6^{N-2} \times 3$ である．これらのモデルをそれぞれ作成し，モデル検査を行うにあたって，3 ラウンドのみをおこなうため，送信元が送信するパケットは，$rreq_S^3$ である．例えば，N が 4 の場合，端末 S，端末 1，端末 2，端末 D が持つ経路表 rt_1，rt_2，rt_D とし，それぞれの取りうる経路表のエントリを，$rt_nm(n \in \{1,2,D\}, m \in \{1,2,3,4,5,6\})$ と表わす．中継端末の経路表のエントリの集合 RT_n を式 (1) と表す．また，宛先端末 D の経路表のエントリの集合 RT_D を式 (2) と表す．

$$RT_n = \{rt_n1, rt_n2, rt_n3, rt_n4, rt_n5, rt_n6\} \quad (1)$$

$$RT_D = \{rt_D1, rt_D2, rt_D4\} \quad (2)$$

各端末の取り得る経路表のエントリの集合 RT_1,RT_2,RT_D となる．ここで，RT_D は，SD ペアの宛先であるため，取り得る経路表のエントリは，$RT_D = \{rt_D1, rt_D2, rt_D4\}$ である．それぞれの経路表のエントリの組合せを作成する．表 3 に示すとおり，その組合せは，$RT_1 \times RT_2 \times RT_D$ となり，端末 ID が 1,2 の端末が 3 ラウンド開始時に取り得る経路表のエントリは，6 通り，宛先端末が取り得る経路表のエントリは，3 通りであるため，組合せの数は，$6^2 \times 3$ である．

本手法では端末が 1,2 ラウンドにおいてどの端末からパケットを受信したかを考慮せず，$rreq, rrep$ をどのラウンドにおいて受け取ったかのみを考慮している．そのため，$rt1$ から $rt6$ に含まれる i,j を具体的な値として扱うことなく，モデル検査を行うことができる．3 ラウンド開始時に各端末がどのような経路表のエントリを保有していたとしても，それらは 3 ラウンドで送受信される パケット $rreq_i^3, rrep_j^3$ から生成されるエントリによって上書きされる．そのエントリの i,j は受信したパケットの送信端末を具体的に表している．$rreq_i^3, rrep_j^3$ を受信しなかった端末は経路構築に関与しないため，$rt1$ から $rt6$ に含まれる i,j がどのような値でもモデル検査に影響を与えない．このように，3 ラウンドにおいては，3 ラウンドで送受信されるパケットのみによって経路が構築されるため，1,2 ラウンドで端末がどの端末からパケットを受信したかを考慮する必要がない．そのため，提案法によって生成される各端末の経路表が取り得るエントリの組合せを表すモデルにおいて i,j を具体的な端末の組合せとして考える必要がない．

表 3: 経路表の取り得るエントリの組合せ

	RT_1	RT_2	RT_D
c1	$rt_1$1	$rt_2$1	rt_D1
c2	$rt_1$2	$rt_2$1	rt_D1
c3	$rt_1$3	$rt_2$1	rt_D1
c4	$rt_1$4	$rt_2$1	rt_D1
...	...	...	...
...	...	...	...
c107	$rt_1$6	$rt_2$6	rt_D2
c108	$rt_1$6	$rt_2$6	rt_D4

3 ラウンドにおいてパケット $rreq_i^3, rrep_j^3$ を送受信し，経路構築に関与する端末は，経路表が保有するエントリの i,j は具体的な端末を示している．経路のループが発生した場合，そのループした経路に含まれる端末が保有するエントリの i,j は具体的な端末を示しているため，本手法によって作成したモデルを検査することによって検出された経路のループは，実際の経路構築において発生するループであるといえる．

5. 実験

本節では，提案方法を用いてモデル検査を行った際の実行結果を示す．ここでのモデル検査は全ての状態を探索するものと，ループを検出することができるかどうかを検証する．比較対象として，1 ラウンドから 3 ラウンドまでをすべて行う場合を挙げる．

5.1. 状態空間の全探索

端末数は 4，SD ペア数は 1，隣接端末数は最大 2 である．端末数が 4 であるため，送信元 1，中継端末 2，宛先 1 である．よって，経路表のエントリの組合せは，$6^2 \times 3$ の 108 通りである．端末は，多くとも 1 回しかクラッシュしない．提案法の実験結果は，108 通り経路表の組合せを表した各モデル検査の実行時間，メモリ使用量，状態数の総和である．また，括弧内は，その平均を表している．また実験環境は以下の通りである．

OS: CentOS6.6　　CPU: Xeon E5-2665
MEM: 128GB　　Tool: SPIN6.2.5

実験の結果を表 4 に示す．1 回目から 3 回目までを 1 度のモデル検査で行う方法は，メモリを使い果たしたため，最後まで状態を探索できていない．提案法によるモ

表 4: 端末数が 4 の場合の実験結果

	提案法 (平均値)	通常検査
実行時間 [s]	$4.8\times10^4(4.5\times10^2)$	$>2.15\times10^3$
状態数	$1.5\times10^{10}(1.4\times10^8)$	$>4.1\times10^8$
メモリ使用量 [GB]	$3.8\times10^3(3.5\times10)$	$>1.0\times10^2$

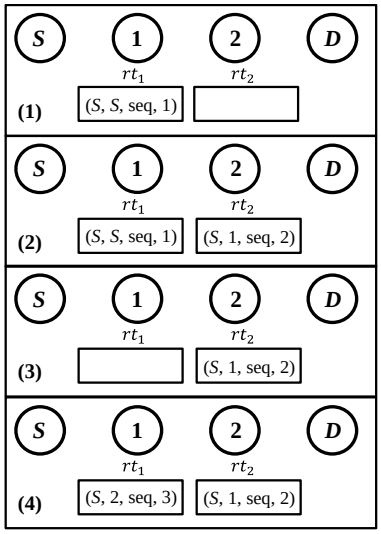

図 7: ループ発生例

デル検査では，108 回のモデル検査全てが実行完了している．1 回のモデル検査に必要な実行時間，状態数，メモリ使用量がそれぞれ削減されている．3 回目の経路構築のみであるため，削減されるのは当然であるが，3 ラウンド開始時のネットワークに存在するパケットを考慮する必要がないため，チャネルに存在するパケットの組合せを表す状態が削減されていることが 1 回のモデル検査における状態数の削減に寄与していると考えられる．また，状態数が削減されたことにより，実行時間，メモリ利用量も合わせて削減されている．

5.2. ループ検出

AODV では，ある端末がクラッシュした後，経路表が空の状態でネットワークに参加することによるループの発生が知られている．例えば，文献 [11] では，端末数が 3 の直線状のトポロジにおいて，経路のループが発生するシナリオを示している．本実験においても，ループ発生が検出できるかどうかについて検証を行う．図 7 にループが発生する例を挙げる．

図 7-(1): 端末 1 は，S からの $rreq$ を受信し，経路表にそのエントリを保有している．

図 7-(2): 端末 1 は $rreq$ をブロードキャストし，端末 2 がそれを受信し，そのエントリを保有する．

図 7-(3): 端末 1 がクラッシュし復帰する．

図 7-(4): 端末 2 は $rreq$ をブロードキャストし，端末 1 がそれを受信し，端末 1 にエントリが保有される．

端末 1 が保有するエントリは，S 宛を端末 2 に送信することになるため，端末 2 が D から $rrep$ を受信し，それをユニキャストで，端末 1 へ送信した場合，端末 1 は

```
1: c_code{
2:  int cnt=0,node=PNode->id;
3:  now.loop=0;
4:  while(cnt<RT_LENGTH){
5:   if(now.tables[node].entries[cnt].dest==S){
6:    node=now.tables[node].entries[cnt].next;
7:    if(node==S){break;}
8:    else if(node==PNode->id){
9:     now.loop=1;break;}
10:   else{cnt=0;}
11:  }
12:  else{cnt++;}
13:  }
14: };
15: assert(loop==0);
```

図 8: ループ検出のために埋め込んだ C 言語コード

$rrep$ を端末 2 にユニキャストで送信することになる．

この状況が発生するかどうかを検証するにあたって，Promela コードに図 8 のコードを加える．図 8 は，ある端末から送信元へのエントリの次ホップを辿ると，自身に戻ってくるかどうかを検出するために追加する．図 8 は，図 3 の中で経路表の操作が終わりパケットを転送する直前である 18 行目と 19 行目の間に配置する．図 8 の変数 node は端末の id を表している．now.loop はループが検出されると 1 になる．now.tables[node] は端末 id が node の経路表を表している．5 行目からは，経路表に宛先 S 宛のエントリがある場合の処理である．6 行目はそのエントリの次ホップを node に代入し，7 行目は，それが S であれば，宛先まで経路が存在することを表している．8，9 行目は，node が自身であれば，ループが発生しているとみなして，now.loop に 1 を代入する．now.loop が 1 になると 15 行目において，loop==0 が偽になるため，この時点でモデル検査が終了する．提案方法によって生成したモデルに図 8 のコードを埋め込み SPIN を実行すると，生成した全てのモデルにおいて Assertion Violation が発生した．提案法によって生成されたモデルは，ループの検出が可能であることを示した．

6. 関連研究

SPIN を用いたモデル検査

[11] では，モデル検査器 SPIN を用いて，AODV のモデル検査を行っている．その中で，ループする経路が生成される可能性があることを明らかにしている．現在のバージョンではその不具合は解消されている．[12] では，WARP を対象にモデル検査を行っている．端末数が 5 であるネットワークの内，2 つの端末の隣接端末数を固定し，*supertrace*/*bitstate* モードを用いて，各端末間でパケットが到達するかどうかの検証を行っている．[13] では，LUNAR を対象に SPIN と UPPAAL を用いてモデル検査を行っている．SPIN を用いたモデル検査では，端末数が 4 から 7，6 つのトポロジを対象にメッセージが正しく到達すかどうかについて検証を行い，その内 2 つは状態の全探索を行えていない．UPPAAL を用いたモデル検査では，端末数が 4 から 7 の 8 つのトポロジを対象に，デッドロックが発生しないこと，経路が構築されること，メッセージが到達していることの検証を行っている．[14][15] では，LAR[16]，DREAM[17]，OLSR を対象にしている．パケットの処理を行う内部の振舞と

隣接端末とのパケットのやり取りを表す外部の振舞を分離してモデル検査を行う手法について述べている．そのなかで，LAR，DREAM，OLSR の知られた不具合について検出を行っている．[18] では，OLSR を対象にモデル検査を行っている．全ての端末への経路が構築されること，MPR を経由することで全ての端末への経路が存在することを検証している．また，端末数が 4 のネットワークにおいて，1 つの端末が誤ったパケットを送信するようなビザンチン故障のモデルも検査している．端末の故障が検出可能なトポロジと検出不可能なトポロジが存在することを明らかにしている．

UPPAAL[19] を用いたモデル検査

[20] では，経路表のエントリの有効期間に注目して，データの通信中にエントリの有効期間が切れるかどうかの検証を行っている．端末数が 12 のネットワークを対象に検証を行った結果，推奨されるパラメータ値では，経路が存在しても経路が構築されないことが示されている．[21] では，LUNAR を対象にモデル検査を行っている．経路が構築できるか，メッセージが正しく到達するかについて検証を行っている．また，状態空間削減のため，Propagating Localized Broadcast with Dampening (PLBD) を提案している．[22] では，AODV は常にホップ数が最小の経路を構築するとは限らないこと，送信元が 2 つ，宛先が 1 つのとき，2 つの経路が構築されない可能性があることを示している．[23] と [24] はモデル検査器として，SMC-UPPAAL[25] を用いている．AODV と DYMO[26] に対して，経路構築のモデル検査を行っている．

本論文では，端末が保有する経路表のエントリに注目し，エントリの取り得る値を組み合わせることでモデル検査を行っている．MANET プロトコルのモデル検査に関する研究は，経路構築の試みを全て行うか，具体的な試行回数を明示していない．複数回の経路構築の試みのモデル検査を 1 回の経路構築のモデル検査のみで行うことが可能であることを示している点において，提案法は，既存の研究と異なるものである．実装されたソースコードを対象にした [27] は，C 言語を入力とするモデル検査器 CMC を用いて AODV の 3 種類の実装 AODV-UU[28]，kernel-AODV[29]，mad-hoc[30] を対象にモデル検査を行い，ソースコードの不具合を検出している．この手法が検出する不具合は，ソースコードを対象にしているため，提案手法とは対象が異なる．

7. まとめ

本研究では，AODV のモデル検査を行う方法を示した．経路構築を 3 回試みる AODV の経路構築のモデル検査を 1 度の実行で検査することは，状態数，実行時間ともに非常に大きくなる．そこで，経路表のエントリに注目し，取り得る経路表の組み合わせをすべて網羅することにより 3 回目の経路構築だけをモデル検査することで，AODV の経路構築を検査できることを示した．経路表の組み合わせ毎にモデル検査を行うため，提案法を用いた場合にはモデル検査の実行回数は増加するが，1 回のモデル検査実行時の状態数，実行時間が短くなること，提案法によって構築されたモデルも経路のループを検出することが可能であることを示した．経路表の組み合わせを個別にモデル検査を行うことが可能であるため，すべての経路表の組み合わせを実行するために必要な時間は，複数の計算機上でそれぞれの経路表の組み合わせをモデル検査することにより，短くすることが容易である．

今後の課題は，本手法を他のプロトコルに適用することである．AODV を拡張したマルチパスプロトコルである AOMDV やテーブル駆動型のプロトコルが対象の 1 つである．また，本手法は，時間を考慮していないため SPIN を用いたが，プロトコルによっては時間を考慮する必要があるため，本手法を時間を考慮した場合のモデル検査に適用する方法も必要であると考えている．

参考文献

[1] A. Boukerche, B. Turgut, N. Aydin, M.Z. Ahmad, L. Bölöni, and D. Turgut, “Routing protocols in ad hoc networks: A survey,” Computer Networks, vol.55, no.13, pp.3032–3080, 2011.

[2] E. Alotaibi and B. Mukherjee, “A survey on routing algorithms for wireless ad-hoc and mesh networks,” Computer Networks, vol.56, no.2, pp.940–965, 2012.

[3] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc on-demand distance vector (aodv) routing,” IETF RFC3561, July 2003.

[4] D. Johnson, Y. Hu, and D. Maltz, The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4, IETF RFC4728, Feb. 2007.

[5] T. Clausen and P. Jacquet, Optimized Link State Routing Protocol (OLSR), IETF RFC3626, Oct. 2003.

[6] R. Ogier, M. Lewis, and F. Templin, Topology Dissemination Based on Reverse Path Forwarding (TBRPF), IETF RFC3684, Feb. 2004.

[7] F. Li and Y. Wang, “Routing in vehicular ad hoc networks: A survey,” IEEE Vehicular Technology Magazine, vol.2, no.2, pp.12–22, June 2007.

[8] H. Hartenstein and K.P. Laberteaux, “A tutorial survey on vehicular ad hoc networks,” IEEE Communications Magazine, vol.46, no.6, pp.164–171, June 2008.

[9] G.J. Holzmann, “The model checker spin,” IEEE Transactions on Software Engineering, vol.23, no.5, pp.279–295, 1997.

[10] H. Kojima, Y. Nagashima, and T. Tsuchiya, “Model checking techniques for state space reduction in manet protocol verification,” Proceedings of International Parallel and Distributed Processing Symposium 2016 Workshop, to appear.

[11] K. Bhargavan, D. Obradovic, and C.A. Gunter, “Formal verification of standards for distance vector routing protocols,” J. ACM, vol.49, no.4, pp.538–576, July 2002.

[12] R. deRenesse and A.H. Aghvami, “Formal verification of ad-hoc routing protocols using spin model checker,” Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference 2004, vol.3, pp.1177–1182, May 2004.

[13] O. Wibling, J. Parrow, and A. Pears, “Automatized verification of ad hoc routing protocols,”

Formal Techniques for Networked and Distributed Systems FORTE 2004, Lecture Notes in Computer Science, vol.3235, pp.343–358, 2004.

[14] D. Câmara, A.A.F. Loureiro, and F. Filali, “Methodology for formal verification of routing protocols for ad hoc wireless networks,” Proceedings of the Global Telecommunications Conference, IEEE GLOBECOM ’07, pp.705–709, Nov. 2007.

[15] D. Câmara, AntonioA.F. Loureiro, and F. Filali, “Formal verification of routing protocols for wireless ad hoc networks,” Guide to Wireless Ad Hoc Networks, pp.189–210, Computer Communications and Networks, Springer London, 2009.

[16] Y.-B. Ko and N.H. Vaidya, “Location-aided routing (lar) in mobile ad hoc networks,” Wireless Networks, vol.6, no.4, pp.307–321, 2000.

[17] S. Basagni, I. Chlamtac, V.R. Syrotiuk, and B.A. Woodward, “A distance routing effect algorithm for mobility (dream),” Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking, pp.76–84, MobiCom ’98, ACM, New York, NY, USA, 1998.

[18] M.F. Steele and T.R. Andel, “Modeling the optimized link-state routing protocol for verification,” Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium, pp.35:1–35:8, TMS/DEVS ’12, Society for Computer Simulation International, San Diego, CA, USA, 2012.

[19] K.G. Larsen, P. Pettersson, and W. Yi, “Uppaal in a nutshell,” International Journal on Software Tools for Technology Transfer, vol.1, no.1-2, pp.134–152, 1997.

[20] S. Chiyangwa and M. Kwiatkowska, “A timing analysis of aodv,” Formal Methods for Open Object-Based Distributed Systems, Lecture Notes in Computer Science, vol.3535, pp.306–321, 2005.

[21] O. Wibling, J. Parrow, and A. Pears, “Ad hoc routing protocol verification through broadcast abstraction,” Formal Techniques for Networked and Distributed Systems FORTE 2005, Lecture Notes in Computer Science, vol.3731, pp.128–142, 2005.

[22] A. Fehnker, R. vanGlabbeek, P. Höfner, A. McIver, M. Portmann, and W.L. Tan, “Automated analysis of aodv using uppaal,” Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol.7214, pp.173–187, 2012.

[23] P. Höfner and A. McIver, “Statistical model checking of wireless mesh routing protocols,” NASA Formal Methods, Lecture Notes in Computer Science, vol.7871, pp.322–336, 2013.

[24] A.D. Corso, D. Macedonio, and M. Merro, “Statistical model checking of ad hoc routing protocols in lossy grid networks,” NASA Formal Methods, Lecture Notes in Computer Science, vol.9058, pp.112–126, 2015.

[25] A. David, K.G. Larsen, A. Legay, M. Mikučionis, and Z. Wang, “Time for statistical model checking of real-time systems,” Computer Aided Verification, Lecture Notes in Computer Science, vol.6806, pp.349–355, 2011.

[26] C. Perkins, S. Ratliff, and J. Dowdell, “Dynamic manet on-demand (aodvv2) routing draft-ietf-manet-dymo-26,” IETF, Feb. 2013. https://tools.ietf.org/html/draft-ietf-manet-dymo-26

[27] M. Musuvathi, D.Y.W. Park, A. Chou, D.R. Engler, and D.L. Dill, “Cmc: A pragmatic approach to model checking real code,” SIGOPS Oper. Syst. Rev., vol.36, no.SI, pp.75–88, Dec. 2002.

[28] “AODV-UU”. http://sourceforge.net/projects/aodvuu/

[29] “kernel-AODV”. http://w3.antd.nist.gov/wctg/aodv_kernel/

[30] “mad-hoc”. http://web.archive.org/web/20010401143715/http://mad-hoc.flyinglinux.net/