M-065

P2P 仮想ネットワークにおける移動体接続の永続化プロトコルとモデル検査

A Realization and Model Checking of the Persistence Protocol for Mobile Connections in P2P Virtual Network

呉 ヒョク † 和崎 克己 †
Hyouk Oh Katsumi Wasaki

1 まえがき

近年，TCP/IP レイヤ上でのオーバーレイネットワークを実現するために，P2P の仕組みを利用して仮想接続を行う事例が増えてきている [1]．実際の接続ノードについて，移動体端末によるネットワーク利用を想定した場合，端末機器の移動による IP アドレスの動的な変更や，下位レイヤ通信の一時的な途絶に伴って，接続を復旧する必要がある．その際，接続を再開するためのコストが増大し，移動する状況によっては多くのオーバーヘッドが発生する問題がある．このため，再接続のコストが低い，新たなコネクション永続化のための上位プロトコルについて検討した．

永続化の仕様が保証されていることを確認するために，SPIN を利用したプロトコル検証 [2][3][4] を行う．SPIN は 1980 年代にベル研究所で開発されたモデル検査ツールであり，分散システムの形式的な検証を行うためのソフトウェアパッケージとして，広く用いられている．SPIN 上でのモデル検査を実行するために，PROMELA(PROcess Meta LAnguage) 言語によってプロトコルとその周辺のモデルを記述する．モデル検査では，初期状態からモデルが取り得る状態遷移を網羅的に生成・探査し，仕様に含まれない不正な状態が発生しないかどうかの検証を，自動的に行う．SPIN は様々な分散システムの検証に用いられており，例えば NAT 越え通信プロトコルの検証に用いられるなどの事例がある [5]．

本研究では，再接続の際，発生するオーバーヘッドを少なくするために，永続化プロトコルについて検討した．設計したプロトコルは，形式検証系 SPIN/PROMELA によって記述し，動作の検証を行った．

2 永続化プロトコルのモデル

図 1 に，本研究で提案している永続化プロトコルの各ノードにおける状態遷移図を示す．

永続化プロトコルは再接続の際，接続を初期状態から開始する際に発生するオーバーヘッドを極力少なくする要請によって仕様が作成された．再接続時のコストは，全く未接続の状態から，接続フェーズの最初より開始した場合に増大する．従って，直前の接続とデータ交換を行っていた時点の情報から (端末情報に基づいた) ユニークなノンス情報を生成し，ノード間で交換，確認出来た後に接続フェーズの途中から接続動作を再開する

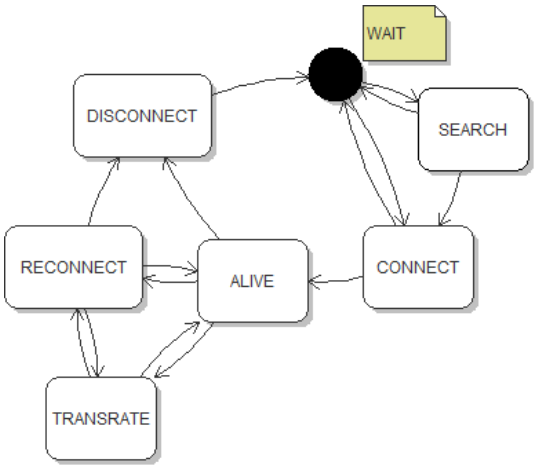

図 1 永続化プロトコルのノード状態遷移モデル

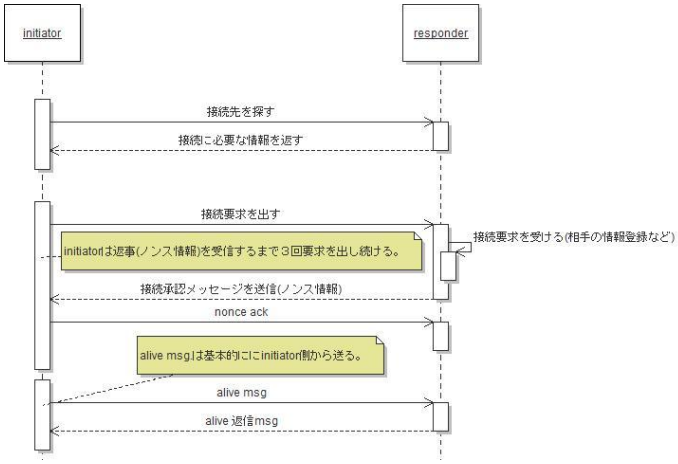

図 2 接続と alive メッセージシーケンス

(RECONNECT)．接続後はノードはお互い接続されていることを確認するために，定期的に接続確認メッセージをノード間で交換する状態 ALIVE に移る．

図 2 に，ノード間で接続を行い，ノード生死の定期的な確認 (ALIVE) や再接続 (RECONNECT) のシーケンスを実施する一例を示す．まず最初にノード initiator は接続先を探す．それに対し，ノード responder は接続に必要な情報を initiator に返す．その後，initiator は responder に接続要求を出し，それに対する接続承認メッセージ (ノンス情報) を受信する．この時，接続承認メッセージを受信出来なかった場合，initiator はこの動作を 3 回まで繰り返す．ノンス情報を受信したら，それに対する ack メッセージを responder 側に送信することで，接続は完了したこととなる．responder は接続承認メッ

† 信州大学大学院工学系研究科,
Graduate School of Science and Technology, Shinshu University.

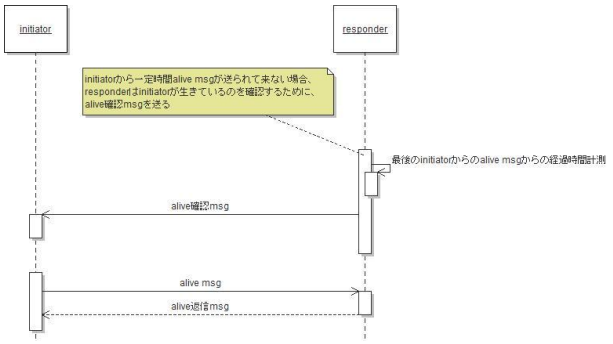

図 3　alive 確認メッセージシーケンス 1

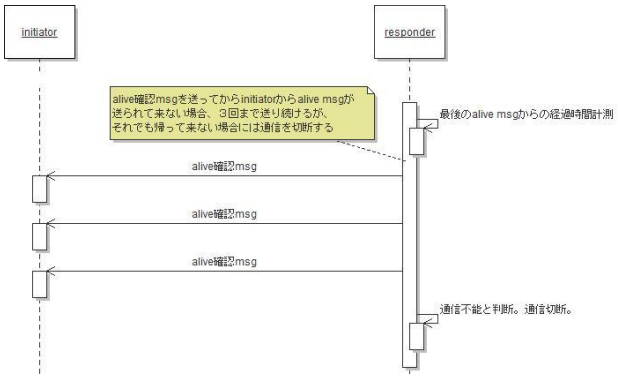

図 4　alive 確認メッセージシーケンス 2

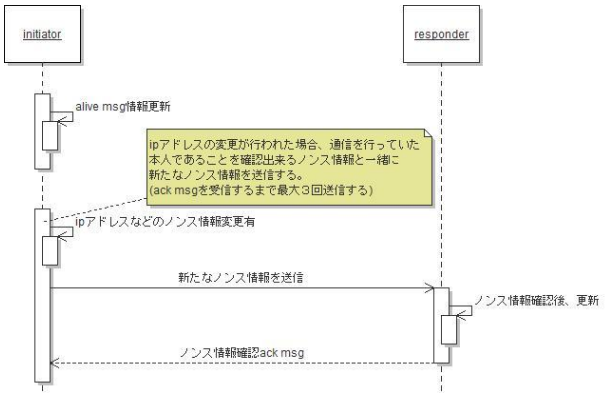

図 5　再接続シーケンス

セージを受信出来たことに対する ack メッセージを受信することで，初めて接続が完了したと判断する．この時，ack メッセージは 1 回しか送信されない．

接続が行われるとノードは alive メッセージの交換を始める．接続要求を出した initiator からは定期的に alive メッセージを送信，これを受信した responder は alive メッセージに対する ack メッセージを送信する．ack メッセージが initiator 側に届かない場合，alive メッセージは最大 3 回まで繰り返し，送信される．

initiator 側から alive メッセージが一定時間内に responder 側に届かない場合，responder は接続生死を確認するために alive 確認メッセージを送信する（図 3）．

alive 確認メッセージも alive メッセージ同様，initiator 側から alive メッセージが送られて来ない場合，最大 3 回まで alive 確認メッセージの送信を繰り返す（図 4）．

どちらかのノード情報（IP アドレスなど）が変更された場合，ノードは reconnect 状態となり，新しいノンス情報とノードを確認出来る情報を一緒に相手側に送信する．新しいノンス情報を受信した側のノードはノンス情報を確認し，更新を行うと，それに対し，ノンス情報確認 ack msg を返す．そして各ノードは直前まで行っていた状態に戻る．この際もノンス情報確認 ack msg を確認出来るまでにノンス情報が変更された側のノードは新しいノンス情報を 3 回まで送り続ける（図 5）．

```
#define ChanLength 2 /* channel length */
#define LossNR 3 /* Border to loss */
#define ChangeNR 3 /* Border to Change */

#define INIT 1 /* initiator ID */
#define RESP 2 /* responder ID */

/* state */
#define WAIT 1
#define ADVERTISE 2
#define CONNECT 3
#define ALIVE 4
#define RECONNECT 5
#define TRANSRATE 6
#define DISCONNECT 7

/* message type */
mtype = {advertise,advertiseAck,nonce,
   nonceAck,alive,aliveAck,checkAlive,
   newNonce,newAck,data,dataAck,disconnect,disAck,none}

/* message structure */
typedef Message{
  mtype mType;
  byte source,dest;
  byte nonce1;
}

/* Protocol structure */
typedef Protocol{
  byte state;
  byte adverAck;
  byte connected;
  byte timer;
}

chan net = [ChanLength] of {Message} /* channel */

Protocol ini,res;

inline rand(number){
  if
    :: number++
    :: number--
    :: number=number+2
    :: number=number-2
  fi;
}
```

図 6　プロセス構造と状態変数の定義

3　提案プロトコルの PROMELA 記述

提案プロトコルの検証のために，プロトコルを仕様記述言語によってモデル化する．提案プロトコルは，図 2,3,4,5 で示したプロトコルの再接続に関する仕様を PROMELA によって記述する．

3.1　プロセス構造と状態変数の定義

最初にチャネルのバッファのサイズやノード ID を定義し，各ノードの状態を定義する．PROMELA 記述の一部を図 6 に示す．ノードの状態は WAIT から DISCONNECT まで定義を行った．そして，ノード間で送受信するメッセージのタイプを定義する．また，LossNR は通信路のパケットロス率に関する定義，ChangeNR はノー

ドのノンス情報が変化する頻度に関する定義である．

メッセージはメッセージタイプ mType とメッセージが送信された送信元 source，メッセージの送信先 dest，そしてノンス情報が格納される nonce1 をメンバーとする構造体で定義する．

ノードプロセスの構造はノードの状態を表わす state，接続先を探す際，ADVERTISE の結果得られた接続先の情報が格納される adverAck，接続されたノードの情報を格納する connected，ノードの振る舞いでメッセージの送信において時間差を置くことで連続して同じメッセージを送信するのを防ぐためのタイマー変数 timer を持つ．

そしてメッセージの送受信で使われるチャネル net を定義し，ノードプロセス構造を持つ変数 ini と res を定義する．また，ノードの中で通信路においてのパケットロスやノンス情報の変更を確率的に行うために乱数生成のための rand（number）マクロを定義する．

3.2 initiator プロセス

initiator プロセスの PROMELA 記述の一部を図 7 に示す．このプロセスは再接続の状態の際，新しいノンス情報の送信を数えるための rcount 変数を持つ．ノンス情報が変更されるきっかけとなる乱数を扱うための変数 cnr と，パケットロス率に関わる乱数を扱うための変数 lnr を持つ．また，メッセージの送受信で利用するために，Message 構造，受信用変数 inM と送信用変数 outM を持つ．そして基本的なメッセージの初期化を行い，プロセスの振る舞いは大きく，メッセージの受信と送信に分割した．

受信はチャネルに initiator 宛のメッセージが存在するかどうか確認を行い，メッセージが存在した場合，受信する．そして受信したメッセージはメッセージタイプによって次の動作を選択する．チャネルに受信するようなメッセージがなかった場合（timeout），プロセスは timer を更新する．そして timer が決められたある定数 (3) に達した時, 受信振る舞いのループから抜けて, 次の送信振る舞いに移る．また，受信の際は乱数 lnr によって通信路のパケットをそのまま捨てることでパケットロス率を表現する．ノンス情報が変更された場合（RECONNECT 状態）にも，通信相手からのメッセージは届かないはずなので，同じようにパケットをそのまま捨てることで表現する．ただし，通信相手に新しいノンス情報を知らせて，それに対する Ack メッセージは捨てない．この Ack メッセージはパケットロスによって捨てられることがある．

送信はノードの状態によって，送るメッセージの種類や振る舞いを分ける．一回送信振る舞いを終えたら，ノードは再び受信振る舞いへと移る (goto RECEIVE). そしてプロセスは乱数 cnr によってノンス情報が変更され，再接続状態に変更される振る舞いを持つ．

ここでは詳しい振る舞い記述は省略する．また，responder プロセスにおいても，基本的に振る舞い記述は initiator 同様である．

4 SPIN を用いた動作検証

PROMELA によって記述されたプロトコルを SPIN モデル検査器によって検証を行った．

```
/* process describing initiator behavior */
proctype Initiator(){
  byte rcount; /* for count sending new nonce msg */
  byte cnr; /* for change nonce info */
  byte lnr; /* for loss msg */
  Message inM, outM;
  d_step{
    outM.source=INIT;
    ini.state=ADVERTISE;
    outM.nonce1=INIT;
    ini.timer=3;
  }
RECEIVE:skip; /* receive behavior */
  do
    :: net?[_,_,INIT,_] ->
       /* receive msg. if its dest is INIT */
       rand(lnr);
       if
   :: ((lnr%5)>LossNR) -> d_step{net?_;}
   :: (ini.state==RECONNECT && !net?[newAck,_,_,_]) ->
      net?_;
   :: else -> net?inM;
       fi;
       if
   :: (inM.mType==advertiseAck) -> /* advertiseAck */
   :: (inM.mType==nonce) -> /* nonce */
   :: (inM.mType==aliveAck) -> skip; /* aliveAck */
   :: (inM.mType==checkAlive) -> /* checkAlive */
   :: (inM.mType==newNonce) -> /* new Nonce */
   :: (inM.mType==newAck) -> /* new Ack */
   :: (inM.mType==data) -> /* data */
   :: (inM.mType==dataAck) -> /* data ack */
   :: (inM.mType==disconnect) -> /* disconnect */
   :: (inM.mType==disAck) -> /* dis ack */
   :: else -> skip;
       fi;
       inM.mType=none;
    :: timeout -> /* retry receive, 3 */
       if
   :: (ini.timer>=3) -> ini.timer=0; goto SEND;
   :: else -> ini.timer++;
       fi;
  od;
SEND:skip; /* send behavior */
  if
    :: (((cnr%5)>ChangeNR) && (ini.state==ALIVE ||
           ini.state==TRANSRATE))
       -> ini.state=RECONNECT;
    :: else -> skip;
  fi;
  if
    :: (ini.state==WAIT) ->
    :: (ini.state==ADVERTISE) ->
    :: (ini.state==CONNECT) ->
    :: (ini.state==ALIVE) ->
    :: (ini.state==RECONNECT) ->
    :: (ini.state==TRANSRATE) ->
    :: (ini.state==DISCONNECT) ->
  fi;
  goto RECEIVE;
}
```

図 7　initiator プロセスの PROMELA 記述（一部）

4.1 ランダムシミュレーション

SPIN は検証において，いくつかの方法がある．まず，ランダムシミュレーションによって，記述した仕様のプロセスを動かし，その振る舞いを出力方法がある．PROMELA で記述した仕様が想定通り記述されていることの確認も兼ねてランダムシミュレーション結果の状態トレースを図 8 に示す．図に向かって左側から initiator，responder ノードの状態を示している．上下方向に向かって時系列を示している．

図 3 のシーケンス図のようにノード間の接続を含めた振る舞いはチャネルのパケットロスやノンス情報の変更で少し変動はあっても，想定通り表現出来ていることが確認出来た．

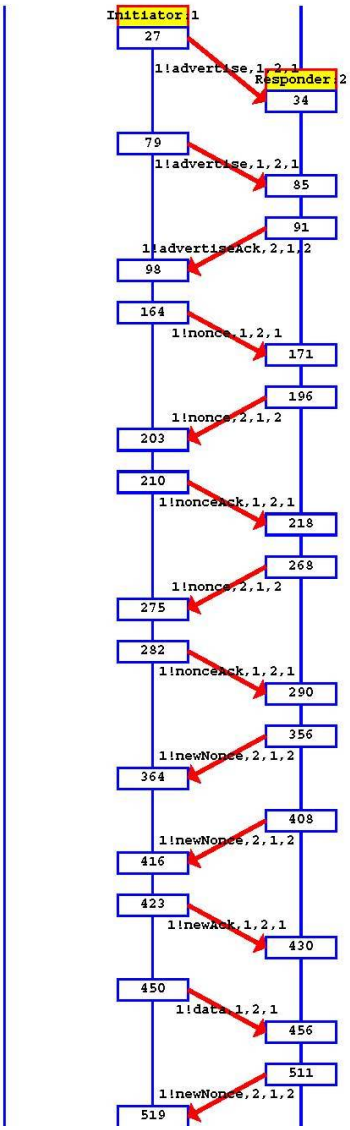

図 8　ランダムシミュレーションの結果

4.2　LTL 式による仕様の検証

SPIN のもう一つの検証方法として，LTL 式を利用して，本プロトコルで守られるべき仕様を定義して，それに反する振る舞いが存在するかどうか確認する検証方法を行う．そのために，まずは本研究で設計したプロトコルで起こってはいけない仕様を考える．

本研究で注目しているノードが再接続（RECONNECT）状態に入った場合，先に図 1 で示した永続化プロトコルの状態遷移モデルで分かるように，次に遷移する状態としてあり得るのは ALIVE 状態と TRANSRATE 状態，そして再接続が失敗した場合に遷移する DISCONNECT 状態 3 つとなる．この仕様をノードプロセス initiator と仮定して，LTL 式で表すと図 9 のようになる．

ただし，この式で表現されている前提条件のみである場合，ノードプロセス initiator が動く前にノードプロセス responder が接続を待ってる状態であり続けるという反例が予想されるため，もう少し仕様を加えるとする．上記の仕様が満足するまで responder が initiator からの

```
(ini.state==RECONNECT) ->
(ini.state==ALIVE || ini.state==TRANSRATE ||
ini.state==DISCONNECT)
```

図 9　LTL 式その 1

```
(res.state!=WAIT) U q
```

図 10　LTL 式その 2

```
(ini.state==RECONNECT) ->
<>(ini.state==ALIVE || ini.state==TRANSRATE)
```

図 11　LTL 式その 3

接続を待ち続ける状態 (WAIT) 以外の状態であるとの仕様を加える，上記の仕様を q だとすると，図 10 のように表すことが出来る．

以上の仕様で検証を行うと本プロトコルで想定されて以内状態遷移はないと確認出来た．

また，ノードが再接続状態に遷移した場合，再接続が成功して，再接続の前の状態に戻ることは存在するのかを LTL 式で検証を行う．この仕様もノードプロセス initiator を仮定して検証を行う．LTL 式は図 11 のように表現出来て，検証を行うと反例は見つからず，成功することが存在すると確認出来た．

5　おわりに

本研究では P2P オーバーレイネットワークにおける仮想接続の永続化プロトコルを提案した．提案プロトコルは，PROMELA 言語でモデル化した後，検査ツール SPIN を用いて検証を行った．提案プロトコルは基本的な要求仕様から離れた動きは見られておらず，再接続も仕様通りに成功されることも確認出来た．

今後は他の確認すべき仕様を考えて，さらに検証を行っていくと同時に，設計したプロトコルを導入した P2P オーバーレイネットワークの構築を行う．

モバイル機器や装置として，組込 μiTRON OS 上のプロトコルスタックとして実装し，peer-to-peer によるマルチホップネットワークを構築し，柔軟な接続性を持つプロトコルとして評価実験などを行う方向で検討していく．

参考文献

[1] 江崎浩, “P2P 教科書”, impress R&D, 2008.
[2] Gerard J.Holzmann, “THE SPIN MODEL CHECKER”, Addison-Wesley, 2004.
[3] Gerard J.Holzmann, “コンピュータプロトコルの設計法”, カットシステム, 2004
[4] 中島震, “SPIN モデル検査”, 近代科学社, 2008.
[5] 西山裕之, 溝口文雄, “モデル検査を用いた NAT 越え通信ソフトウェア検証の研究”, 日本ソフトウェア科学会第 23 回大会論文集, 3C-2,pp.1-5,2006