モデル検査を用いたNAT越え通信ソフトウエア検証の研究

Verification of NAT Traversal Communication Software using Model Checking

西山 裕之 †, 溝口 文雄 †
Hiroyuki NISHIYAMA, Fumio MIZOGUCHI

† 東京理科大学理工学部
Faculty of Sci. and Tech. Tokyo University of Science
nisiyama@ia.noda.tus.ac.jp

本研究では，広域分散メカニズムである NAT 越え技術対して，モデル検査ツールである SPIN を用いて通信ソフトウエアの検証を行った．SPIN は本来，分散システムの形式的な検証を行うためのツールであり，モデルが取り得る状態遷移を網羅的に生成し，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．このような特徴を用いることで，本研究では我々の開発した NAT 越え技術に対するソフトウエア検証を実施し，接続を確立するためのプロトコル内に接続不良を起こす可能性を確認することができた．さらに，その改善方法を導出し，新たに作成したモデルにより検証を行った結果，不具合を解消することに成功した．

1 はじめに

近年，グリッドコンピューティングに代表されるネットワーク計算機間の情報共有や協調処理が注目されており，プライベートネットワーク上の計算機群を用いるための広域分散メカニズムとして NAT(Network Address Translator) 越え技術に注目が集まっている [3]．これにより，近年になり急速に増加した，企業，大学，および一般家庭内に存在するプライベートネットワーク内の計算機による，協調計算やテレビ会議のような情報共有に基づくサービスを広範囲で行えるようになった．しかしながら，このような新しいサービスを実現するメカニズムの設計を行う場合，そのメカニズムの頑健性や安全性を検証することは，非常に重要となる．そこで本研究ではモデル検査ツールである SPIN を用いて通信ソフトウエアの検証を行った．

SPIN[2] は 1980 年代にベル研究所 [6] で開発されたモデル検索ツールであり，分散システムの形式的な検証を行うためのソフトウエアパッケージとして，広く配布されている [4]．SPIN で検証を行うモデルは，PROMERA(PROcess MEta LAnguage) という言語で記述されることにより，SPIN により実行が行われる．この実行により，モデルが取り得る状態遷移を網羅的に生成し，不正な状態（例えば，デッドロックや未定義の受信など）が発生しないかどうかの検証を自動的に行う．以上の特長により，SPIN は様々な分散システムの検証用ツールとして用いられており，1997 年の火星調査に用いられたロボット（Mars Pathfinder）の制御システムの設計にも検証用に用いられた他，セキュリティプロトコルの検証用にも用いられている [4, 5]．

以上のような背景より，本研究では，我々の開発した広域分散メカニズムである NAT 越え技術である LampEye プロトコル [3] の検証を実施した．その結果，接続を確立するためのプロトコル内に，接続不良を起こす可能性を確認することができた．さらに，その改善方法を導出し，新たに作成したモデルにより検証を行った結果，不具合を解消することに成功した．

2 NAT 越え通信プロトコル：LampEye

NAT 越え技術とは，プライベートネットワークに阻まれた計算機どうしの直接的な情報共有を可能にする技術である．NAT 越え問題の解決策として，NAT にポートフォワーディングの設定を行う方法があるが，ネットワーク管理者権限を必要とするなどの制限が生じる．そのため，アプリケーションレベルでの NAT 越えを行う方法として，以下の 3 種類の方法が考えられている．

- UDP hole punching[1]

  NAT の内側にある計算機のポートに対するマッピング情報を用いた通信方法．約 85 %の NAT に適用可能 [1].

- UPnP[7]

NAT に対してポートフォワーディングの設定を自動的に行う通信方法．NAT ルータには高い割合で実装されているが，UPnP 機能を無効にされている場合がある．

- 中継方式

  各 NAT の外側に計算機を用いて中継させる方式．確実に中継が可能となるが，中継させる計算機に通信負荷が集中するため，通信遅延が生じる危険性が存在する．

以上より，複数の異なるプライベートネットワーク間の計算機群を統合的に利用するためには，様々な種類の NAT を介することから，NAT の種類に応じた NAT 越え方法を各計算機間で用いることになる．そこで，広域の計算機間の統合処理を自動化するためには，計算機間で使用する通信プロトコルを決定するための新たな通信プロトコルの設計が必要となる．我々の設計した LampEye プロトコル [3] では，図 1 の様に，UDP hole punching，UPnP の順に接続の可能性を確認した後，いずれでも接続に失敗した場合のみ，通信遅延の危険性のある中継方式を選択している．このように，各計算機間で用いる通信方法の決定時（通信路の確保時）において，煩雑な通信処理を必要とすることから，そのメカニズムの頑健性や安全性を検証することは，非常に重要となる．そこで本研究ではモデル検査ツールである SPIN を用いて通信ソフトウエアの検証を実施した．

3 LampEye プロトコルに対する SPIN による検証

3.1 PROMERA によるモデル化

本研究においてモデル化し検証する NAT 越え通信プロトコルは，図 1 のようになっている．ここで，モデル化するプロセスとして，接続を行う側のプロセス（ノード S）を Initiator プロセス，接続を受ける側のプロセス（ノード R）を Responder プロセス，そして，仲介するプロセス（仲介ノード）を Server プロセスとして個別に定義を行った．各プロセスの持つ変数は次の通りである．

```
typedef Initiator { /* initiator state */
  byte state;
  byte natType, usePort, natLenge;
  byte resNatLenge;
  byte responderPort}

typedef Responder { /* responder state */
  byte state;
  byte natType, usePort, natLenge;
  byte iniNatLenge;
  byte initiatorPort}

typedef Server {   /* server state */
  byte state;
  byte drIni, drRes}
```

変数 state は各プロセスの状態を表し，変数 natType, usePort，natLenge は，それぞれ，NAT のタイプ（hole punching が可能かどうか等の情報），使用するポート番号，hoke punching におけるマッピングポートの情報である．変数 resNatLenge，iniNatLenge, responderPort, initiatorPort や変数 drIni，drRes は，Initiator プロセスや Responder プロセスからのマッピングポートやポート番号の情報，Server プロセスにおける接続先の情報である．なお，変数 natType, usePort, natLenge の値は，様々なタイプの NAT の利用や環境の変化に対応するために，検証時は複数の選択肢が存在するものとする．

また，各プロセス間で行われる通信メッセージは次のように定義を行った．

```
typedef Message { /* message format */
  byte to,from;
  mtype mType;
  byte data1,data2}
```

変数 to, from はメッセージの送受信者の情報を，変数 mType はメッセージ種類（ステップごとにメッセージの種類は異なる），そして，変数 data1 と data2 は，各ステップ内で使用されるポート番号やマッピングポートの情報が必要に応じて代入される．

LampEye では通信方式として TCP 通信と UDP 通信の 2 種類を使うことから，次のように 2 つの通信チャネルの定義を行った．

```
chan tcp = [ChanLength] of {Message}
chan udp = [ChanLength] of {Message}
```

これは，上記の通信メッセージ Message で定義したメッセージを，各チャネルが変数 ChanLength の数だけデータを保持できることを意味している．なお本プロトコルでは，各チャネルが保持するメッセージ

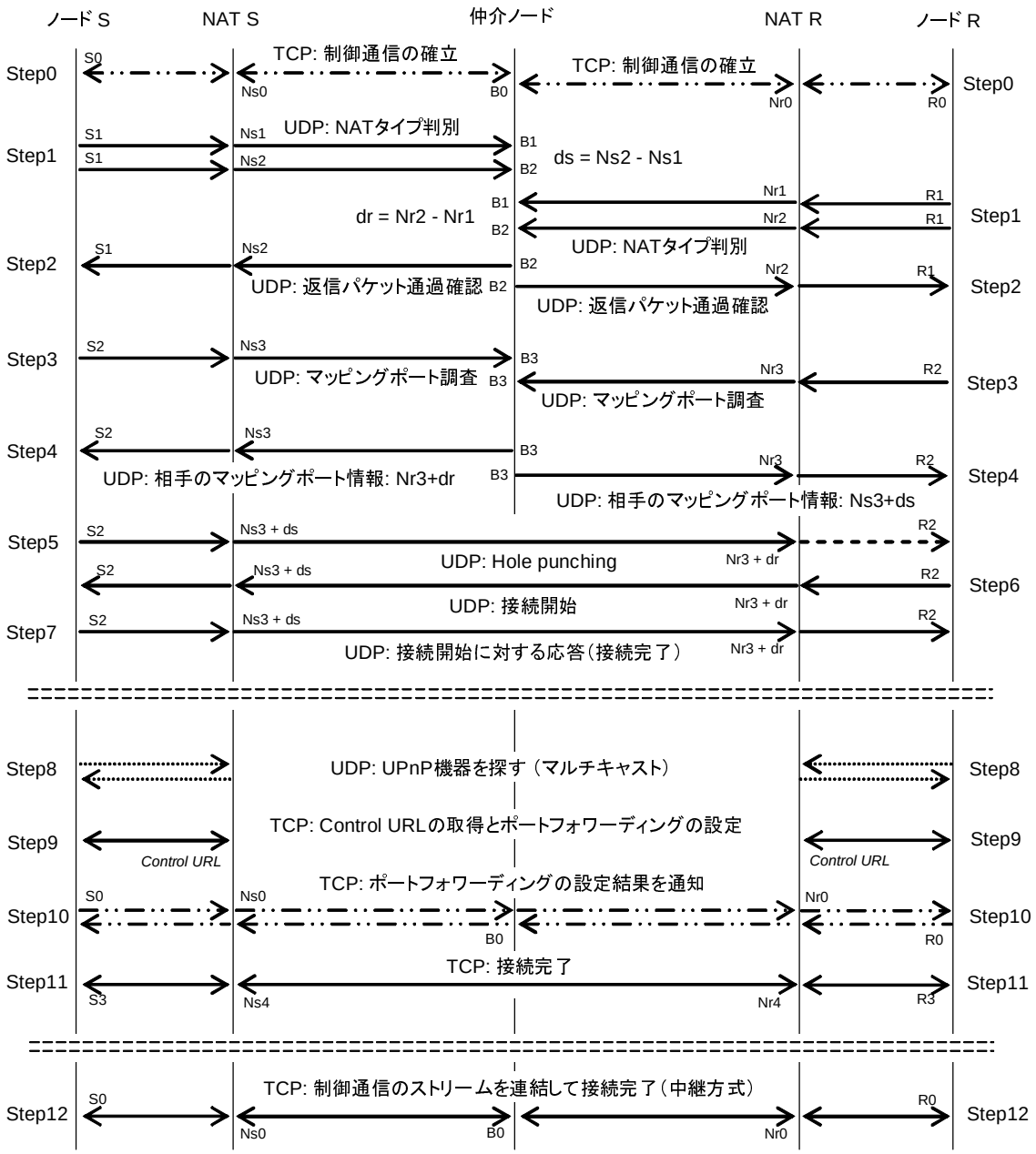

図 1: NAT 越え通信プロトコル (LampEye プロトコル)

数が 4 を超えることは無いので，変数 ChanLength は 4 で固定している．

各プロセスのモデル化は，図 1 のステップごとに定義を行った．Initiator プロセスと Responder プロセスは，各ステップを順番に実行するように定義を行い，Server プロセスは，Initiator および Responder プロセスからのメッセージ受信に応じた処理を行えるように定義を行った．また，Initiator プロセスおよび Responder プロセスにおいて，使用する NAT の種類やポート番号，および使用ポートの差分などの条件を変更可能にするとともに，使用する NAT の種類によっては UDP 接続を断念する定義を行った．このとき，Initiator プロセスにおけるステップ 3 からステップ 6 は，次のように記述できる．

```
Initiator ini;
Message inM, outM;

  /* step3 */
  d_step {
    outM.to = SERV;
    outM.from = INIT;
    outM.mType = msgCheckMap;
    outM.data1 = ini.usePort;
    udp!outM};

  /* step4 */
  if
  :: d_step { udp?[INIT,SERV,msgInfoMap]
                              -> udp?inM};
     ini.responderPort = inM.data1;
     ini.resNatLenge = inM.data2;
  :: else -> goto MULTICAST;
  fi;

  /* step5 */
  d_step {
    outM.to = RESP;
    outM.from = INIT;
    outM.mType = msgHolePunch;
    outM.data1 = ini.responderPort;
    outM.data2 = ini.resNatLenge;
    udp!outM};

  /* step6 */
  if
  :: d_step{
     udp?[INIT,RESP,msgConnectUDP,
          ini.usePort,ini.natLenge]
                           -> udp?inM};
  :: else -> goto MULTICAST;
  fi;
```

ステップ 3 は Responder との間でマッピングポート調査を行うために，中継サーバを介して Initiator のポート番号情報（変数 usePort）を送信している（ステップ 1 の段階でマッピングに必要な情報は中継サーバが保持済）．ここでは，メッセージ変数 outM に，Initiator から Server に対するメッセージとして（変数 from，to に代入），UDP の通信チャネルに送信を行っている（なお，メッセージタイプの megCheckMap はステップ 3 におけるメッセージを意味する）．

ステップ 4 では，中継サーバを介して Responder からのマッピングポートの情報を得るためのメッセージを受け取っている．このメッセージを受け取ることにより，Responder の使用しているポート番号とマッピングポート情報を受け取り，それぞれを Initiator の変数 renponderPort と resNatLenge に代入している．なお，本メッセージが受け取れなかった場合（タイムアウトした場合），自分（Initiator）に対するマッピングポート調査が失敗したものと判断し，hole punching による通信方法を断念し，UPnP 方式による接続を試みることになる．

ステップ 4 により無事に受信処理が完了した後，ステップ 5 とステップ 6 では，得られた情報に基づき，互いに UDP チャネルを用いて直接メッセージの送受信を行い，ステップ 6 で Responder からのメッセージの受信を確認することで，hole punching による情報共有を開始する．なおステップ 7 は，ステップ 6 でメッセージの受信処理に成功すると，相手に接続完了メッセージを送信するだけである．

その他，本プロトコルにおいて，各ステップの受信処理に制限時間が存在し，タイムアウトが生じると接続に失敗したものとして判断が行われる．この結果，ステップ 1〜7 で失敗した場合はステップ 8 へ，ステップ 8〜11 で失敗した場合は，ステップ 12 へ処理が移る．本モデル化では，このようなタイムアウト処理において，UDP 通信における各受信処理の定義において加えることで，タイムアウト後の処理の検証も行えるように作成を行った．

4 SPIN による検証実験

上記で作成したモデルを SPIN により検証を行った結果，Responder プロセスにおいて，全てのステップが終了しない可能性が存在する旨を意味するエラーが提示された．そこで，そのようなエラーとなる状態の追跡を行うと，次のような状態において，上記のエラーが発生することが確認できた（SPIN には，エラーが確認された場合，そのエラーが発生する状態遷移の過程を確認する方法が存在する）．

- Initiator プロセス：ステップ 7 まで無事に終了

し，UDP による接続を完了させた状態

- Responder プロセス：ステップ 7 においてタイムアウトが発生し，UPnP 接続を行う過程において TCP 受信待ち状態で停止している．

以上より考察できるプロトコルの不具合として，ステップ 7 において，Initiator プロセスは接続完了メッセージを送信して UDP 接続に基づく通信状態を確立させたものとして通信処理に移行するが，Responder プロセスでは，その完了メッセージの受け取りに失敗することでタイムアウトとなり，次の接続方法であるステップ 8 以降に接続処理を移すことに成る．実際問題として，本エラーはステップ 5 で Initiator プロセスから Responder プロセスへの UDP 通信が成功した後に生じることになるから，その可能性は低いものとも考えられるが，ステップ 4 で受け取ったマッピング情報に誤りが存在した場合，ステップ 5 が成功したとしても，ステップ 7 が成功するとは限らない．そのため，本エラーが生じる可能性が存在することを検証できたことは，システム上において致命的なエラーの存在を確認できたことになる．

なお，本モデルでは，TCP 受信待ちにおけるタイムアウトは導入していないため，上記のようなエラーが検出されたが，導入していた場合，Initiator プロセスが UDP 接続を行い，Responder プロセスが TCP 接続を行うというエラーが検証されたものと考えられる．実際に，広域分散メカニズムにおける研究の一環として，本 NAT 越え技術を用いた大規模情報共有システムの設計を行っているが，ネットワーク環境（NAT の種類）によって，互いに異なる通信形態で接続を試みる不具合も確認されている．

以上より，広域通信メカニズムには，使用する NAT の種類によっては接続に不具合を生じさせるプロトコルエラーの存在が確認されたことになる．そこで，次節では，プロトコルの改善と再検証を実施した．

5 NAT 越えプロトコルの改善

前節で示したプロトコルの不具合は，図 1 におけるステップ 1〜7 の UDP 接続において，接続の完了確認に，タイムアウトの生じる UDP 通信を使用していることが原因として考えられる．

ここで，本メカニズムには，UDP 接続に必要な情報のみを中継する仲介ノードが存在している．そこで，UDP 接続の終了確認を，仲介ノードを介して TCP 通信により行えるようにする改善を考える．具体的には，図 1 のステップ 7 の後に，Initiator および Responder プロセスが仲介ノードである Server プロセスに接続結果の報告を行い，各報告を，Server プロセスから Initiator および Responder プロセスに返信する，というものである（その他，各ステップにおけるタイムアウト時の処理も一部変更している）．この通信の結果，Initiator および Responder プロセスでは，自分が UDP 接続に成功していたとしても，相手の接続状態が確認できることから，前節のようなエラー状態は生じないことになる．

以上の様に，前節で作成したモデルに対して，ステップ 7 の後に，仲介ノードである Server プロセスに接続完了状態の確認するための通信を加えたモデルを作成し，SPIN で再検証を試みた．この結果，エラーは生じることなく，全てのプロセスにおいて正常終了したことが確認された．

その他，SPIN の Never Claim 機能等を用いて，プロセスの正常終了の結果として，異なる通信形式での接続を行ってしまう不具合の確認や，UDP 接続，UPnP 接続，TCP 接続が成立する各状態が存在することも確認した．

6 おわりに

本研究では，広域分散メカニズムである NAT 越え技術 LampEye に対して，モデル検査ツールである SPIN を用いて検証を実施した．その結果，接続を確立するためのプロトコル内に，接続不良を起こす可能性を確認することができた．さらに，その改善方法を導出し，新たに作成したモデルにより再検証を行った結果，不具合を解消することに成功した．

参考文献

[1] B. Ford, P. Srisuresh, D. Legel: Peer-to-Peer Communication Access Network Address Translators, Proc. of USENIX 2005, pp. 179-192, 2005.

[2] G. J. Holzmann: *The SPIN MODEL CHECKER*, Addision Wesley, 2004.

[3] 大迫勇哲，山崎航，西山裕之，溝口文雄：透過的なネットワーク環境を実現するグリッドミドルウエア，日本ソフトウエア科学会第 22 回大会論文集，6C-1, 2005.

[4] 田中慎也，佐藤文明，水野忠則：SPIN に基づくセキュリティプロトコル検証システム，情報処理学会論文誌，Vol. 42, No. 2, pp. 147 - 154, 2001.

[5] W. Wen, F. Mizoguchi and Y. Al-Salqan: Model Checking Security Protocols: A Case Study Using SPIN, 東京理科大学情報メディアセンターテクニカルレポート，AIST-TR98-033, 1998.

[6] BELL Laboratories, http://www.bell-labs.com/

[7] Universal Plug and Play, http://www.upnp.org/