The Institute of Electronics, Information and Communication Engineers

モデル検査を用いた通信プロトコル二重化の検証

池田　聡[†a)]　地引　昌弘[†]　久野　靖[††]　西森　丈俊[††]

Verification of Dual Redundant Communication Protocols Using Model Checking

Satoshi IKEDA[†a)], Masahiro JIBIKI[†], Yasushi KUNO[††], and Taketoshi NISHIMORI[††]

あらまし　高い信頼性を求めるサービスに対して高可用性を確保するために，フェイルオーバクラスタが利用されている．このようなシステムでは，フェイルオーバ時にクラスタと通信相手が継続した通信を行うために，プロトコルスタックの内部状態を同期する必要がある．しかし，並行動作するマシンの数が増えると，同期機構の検証作業が困難になるという問題が生じる．そこで，本論文ではクラスタの同期機構を設計段階で検証するため，モデル検査を利用した通信プロトコル二重化の検証手法を提案する．提案手法では，モデル検査を用いることにより，設計段階における不具合の早期発見が可能である．加えて，単体システムのモデルとその形式仕様をクラスタの検証に活用することにより，通信プロトコル二重化の検証を汎用的な手順で行うことができる．提案手法をTCPの二重化に対して適用し，提案手法の有効性を確認した．

キーワード　モデル検査，通信プロトコル，冗長化

1. まえがき

ネットワークサービスの多様化に伴い，サーバやネットワークに高い可用性を求めるサービスが増えている．企業の基幹業務にかかわるシステムや，ネットワーク経由でソフトウェアの機能を利用するSaaSなど，高可用性が必要なサービスは多岐にわたる．

サーバシステムの高可用性を確保する方法の一つとして，フェイルオーバクラスタがある[1],[2]．フェイルオーバクラスタでは，同じ機能を備えた複数の機器を用いて，一部で障害が発生した場合に他の機器へ処理を切り換えることにより可用性を確保する．この切換をサービス利用者から隠ぺいするために，クラスタ内の機器間ではシステム状態の同期が行われる．クラスタリングに必要な同期の内容は，運用されるサービスにより異なる．例えば，静的なコンテンツのみを配信するWebサーバであれば，同期の必要はない．一方，DBサーバでは，ミラーリングや共有ディスクなどを利用してディスクの内容を同期する必要がある．

サービスによっては，現用–予備間の切換後も，クラスタと通信相手（対向）の通信を継続することが必要となる．これを実現する方法の一つに，現用と予備を同時に動作させながら，予備から対向（外部）への出力処理を現用が故障するまで抑制し，プロトコルスタックに対するイベントを外部から制御することで同期を行う方式がある．[3]ではTCPについて，[4]ではTCPとSCTPの二つのプロトコルについて，この方式に基づいたクラスタリングを行っている．しかし，これらの従来研究では，プロトコル二重化の方式提案とその性能評価を主目的としており，二重化設計に対する妥当性の検証について議論はなされていない．

二重化の検証が十分でないと，イベントの発生における微妙なタイミングの差が原因で，現用と予備の内部状態が厳密に一致しない場合が生じる．その結果，通信の継続に必要な同期がとれない可能性がある．システム開発における最終的なテストの段階で同期のとれないことが判明すると，設計段階まで戻って修正する必要があり，多大なコストが発生する．また，クラスタ化によって並列実行するマシンが増えるとバグの再現性が低くなり，そのトレースが困難になる．このような問題を回避するには，実装後に問題が出ないよ

[†] NEC システムプラットフォーム研究所，川崎市
System Platforms Research Laboratories, NEC Corporation, 1753 Shimonumabe, Nakahara-ku, Kawasaki-shi, 211–8666 Japan
[††] 筑波大学大学院ビジネス科学研究科，東京都
Graduate School of Buisiness Sciences, University of Tsukuba, 3–29–1 Otsuka, Bunkyo-ku, Tokyo, 112–0012 Japan
a) E-mail: s-ikeda@fd.jp.nec.com

 ©（社）電子情報通信学会 2010

NII-Electronic Library Service

The Institute of Electronics, Information and Communication Engineers

うに，設計段階において検証を行うことが重要になる．

プロトコルの設計段階における検証は，[5], [6] など多くの事例が報告されているが，プロトコルの二重化を対象とした検証事例は，筆者らの知る限りでは報告されていない．更に，プロトコルの二重化方式については，基本的な設計方針が存在するものの，その検証方法に対する枠組みはない．そのため，プロトコルごとに検証方法を検討する必要があり，プロトコル二重化の設計検証は困難である．

本論文では，次のアイデアに基づく通信プロトコル二重化の検証手法を提案する．

- モデル検査を利用した設計段階の検証
- 単体系における形式仕様の冗長系への適用
- 検査済み単体系を活用した冗長系モデルの作成

以下，**2.** では通信プロトコルの二重化手法とその問題点について述べ，**3.** ではモデル検査を用いた二重化の検証手法について説明する．**4.** では，提案手法をTCP に適用した結果について述べる．**5.** では提案手法の有効性について考察し，最後に，**6.** でまとめる．

2. 通信プロトコルの二重化とその問題

2.1 二重化手法

本論文で検証対象とする二重化手法は，プロトコルスタックの内部に修正を加えず，外部からのイベント制御（後述）により内部状態を同期する．一般に，プロトコルスタックは，データの受信やタイマーなどのイベントを受けて内部状態を変化させるイベント駆動型で動作する．そこで，これらイベントをプロトコルスタックの外部で制御することにより，二重化したプロトコルスタックの現用–予備間で内部状態を間接的に同期させることが可能となる．このような二重化は，現用が単体で動作するシステム（単体系）に基づいて，現用と予備からなるクラスタ（冗長系）として構成される．図 1 は，このような二重化構成を模式的に表したものである．

図 1 に示すように，冗長系では，現用・予備ともに単体系における現用のプロトコルスタックをそのまま利用する．そして，対向から冗長系への通信は現用と予備の両方に配送され，通常時は現用から，現用の故障時は予備からの通信のみが対向へ届くよう制御される．ただし，これだけでは現用と予備のプロトコルスタックに状態の不一致が生じるため，スタックへの入力の直前及びスタックからの出力の直後で同期機構を動作させて不一致を抑制する必要がある．このような

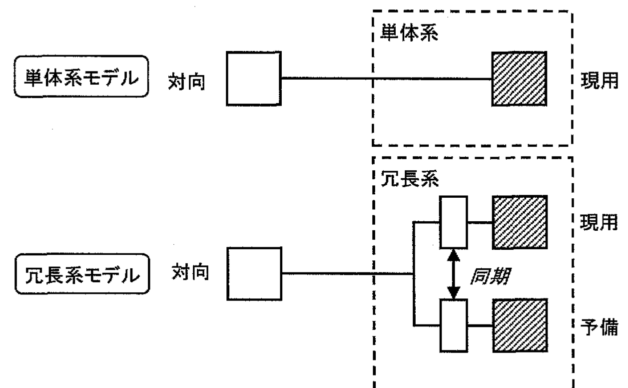


図 1 プロトコル二重化
Fig. 1 A dual redundant protocol.

不一致は，例えば現用と予備に届くイベントの微妙なタイミングの差や順序の違い，あるいは一方におけるイベントの損失などが原因で生じる．

同期機構の役割は，現用/予備における内部状態の不一致に起因する致命的な不具合を防ぐことにある．同期機構は，例えば，現用と予備における情報交換，送受信のタイミングの制御，メッセージの書換えなどを用いて同期を行う．ここでの同期は，現用と予備の内部状態を厳密に一致させることではなく，不具合が発生しない許容範囲内に内部状態の不一致を抑えることを指す．プロトコル二重化において不具合が発生するのは，現用/予備の内部状態が大きく乖離した状態で，フェイルオーバ処理が起こる場合である．現用–予備間で内部状態の違いが大きいと，同じイベントを受信してもその後の振舞いが一致しないため，結果としてフェイルオーバ後に対向–予備間の通信が継続できなくなるからである．したがって，同期機構の設計では，現用と予備の動作が一致しない原因となる内部状態の乖離を特定し，それを抑制する必要がある．

2.2 問 題 点

前節で述べたとおり，通信プロトコルの二重化を実現するには，致命的な不具合の原因となる現用/予備における内部状態の不一致を特定しなければならない．しかし，この原因究明を阻害する，以下の課題が存在する．これらの課題は，単体系の検証においても問題となるが，冗長系では特に顕著となる．なぜなら，冗長系では現用と予備の並列動作によりシステムの動作がより複雑になるためである．

発見の遅れ： 同期機構の検証を実装後のテスト段階で行う場合，不具合の原因が設計段階にあったとしても，テストを実施するまでその発見が遅れることになる．そのため，設計段階の不具合が判明すると手戻

NII-Electronic Library Service

The Institute of Electronics, Information and Communication Engineers

りが発生し，開発の遅れやコストの増加を招く結果となる．また，テストでは設計と実装の両段階に起因する不具合が発見される可能性があり，その切分けが難しくなる [7]．

再現性の低さ： 複数ノードが並列に動作するシステムでは，タイミングの違いにより，実行ごとに結果が異なる可能性がある．そのため，不具合がタイミングに依存する場合は，その再現性が低くなる．特に，本論文で対象とする二重化手法では，現用と予備でほぼ同じ動作をしながら処理が進むため，両者の微妙なタイミングの違いが不具合の原因となることが多く，テスト時にその微妙な違いを再現させることは難しい．

追跡の困難さ： 不具合が発生した場合に，その原因を追跡するには，現用と予備におけるプロトコルスタックの内部状態を照合して送受信データを監視し，これらのイベントを適切なタイミングで記録しなければならない．これらの実行記録は，パケットの送受信や状態遷移のたびに蓄積されるため膨大な量になる．そのため，不具合の原因となる現用と予備の動作に生じた不一致を突き止めることは，非常に困難な作業となる．

これらの問題を解決する有効な方法として，まず，開発プロセスの初期段階（例えば，設計段階）において検証を行うことが考えられる．検証方法は，再現性の低い不具合でも検出できるように，動作のタイミングに依存しない網羅的な手法でなければならない．更に，不具合の分析が容易に行えるように，不具合となる実行系列が簡潔に出力される必要がある．

以上に加え，プロトコル二重化では，二重化を行うプロトコルごとに検証方針を検討しなければならないという課題がある．これは，プロトコル二重化に対する汎用性のある検証手法が確立されていないためである．これに対し，プロトコルの二重化手法には，**2. 1** で述べたようなプロトコルに依存しない基本的な設計方針が存在することから，この方針に基づく二重化プロトコルを，汎用的な手法により検証できることが望ましい．

3. 二重化の検証手法

二重化したプロトコルでは，通信中に現用で故障が発生したとしても，対向がそれに気づくことなく処理を継続できることが目標となる．多くのプロトコルは，実装による動作の違いを許容できるように設計されているため，対向が気づかなければ，対向から見た冗長系の動作が，単体系の動作と必ずしも同一である必要はない．そこで本論文では，単体系の満たすプロトコル仕様が冗長系でも満たされた場合にプロトコルが二重化されたと定義し，以下ではこの定義に基づいた冗長系の妥当性を確認する手法について述べる．

プロトコルの性質を検証するには，モデル検査 [8] を利用する方法が広く用いられており，これはプロトコル二重化の検証にも利用できる．モデル検査による検証では，モデル記述言語で記述したプロトコルの状態遷移モデルが，形式仕様を満たすかどうかを網羅的に検証する．形式仕様として，例えば，本論文で利用するモデル検査器 SPIN [9] では，LTL（Linear Temporal Logic）式が利用できる．単体系の仕様は，対向と現用の内部状態を参照する LTL 式として記述できる．モデル検査を利用することにより，**2. 2** で挙げた課題を解決できる．

図 1 で示す二重化構成の冗長系では，現用が故障しない限り，対向と現用間の通信は単体系モデルにおける対向と現用の通信と同等であり，単体系で成り立つ性質は冗長系でも成り立つ．また，予備の故障時には，現用の内部状態は対向との通信状況と整合性がとれており，現用は同期機構を用いない従来どおりの動作を行えばよい．したがって，検証が必要となるのは，現用が故障する場合における対向–予備の動作である．したがって，単体系モデルの性質を対向–予備間の性質として修正した LTL 式を利用することで，冗長系モデルを検証できる．これより，単体系モデルから冗長系モデルを構成することで，二重化検証における汎用性を実現できると考えられる．

以上を踏まえ，本論文では以下の手順によるプロトコル二重化の検証を行う．

1. 単体系モデルの作成と事前検査：
   冗長系モデルを作成する前に単体系の仕様を確認する．
2. 同期なし冗長系モデルの作成：
   単体系モデルに基づいて，同期機構を含まない冗長系モデルを作成する．
3. モデル検査による反例の抽出：
   冗長系モデルのモデル検査を実施し，同期機構設計のヒントとなる反例を抽出する．
4. 反例の分析と同期機構の導出：
   抽出した反例から現用と予備の重大な不一致の原因を分析し，同期機構を導入する．

本論文で二重化検証の対象とするプロトコルは，メッ

NII-Electronic Library Service

The Institute of Electronics, Information and Communication Engineers

セージの順序反転及びその損失が生じる通信路で 1 対 1 の通信を行うプロトコルである．特に，ノード同士が互いの状態を逐次把握するステートフルなプロトコルを対象とする．これは，互いの状態を保持する必要がなければ，冗長化の際に同期の必要が生じないためである．一般にステートフルなプロトコルのメッセージ交換では，処理の進捗を管理するため，メッセージごとに識別子を付与し，確認応答を用いてメッセージの到達を確認する．メッセージの順序がプロトコルスタックの動作に重要な意味をもつ場合は，メッセージの識別子としてシーケンス番号が広く用いられる．

以下では，シーケンス番号を利用するプロトコルとして，スライディングウィンドウ方式を利用した通信プロトコル（以下，SWP: Sliding Window Protocol）の一般的なモデルを定義し，これを用いて提案手法を説明する．

3.1 単体系モデルの作成と事前検査

単体系の性質について確認するため，モデル記述言語 Promela で SWP の単体系状態遷移モデルを定義する（付録参照）．なお，モデルが煩雑になり可読性が低下するのを防ぐため，付録ではメッセージの到着順序の反転はモデル化していないが，実際の検証は順序反転を導入したモデルを用いて行っている．単体系モデルは，双方向の通信チャネルによって通信する二つのプロセスとして記述する．各プロセスは，イベントループをもち，メッセージ受信や再送などのイベントを契機に内部状態の変更及びメッセージ送信を行う．

モデルの性質は，プロセスの局所変数を参照する LTL 式として記述される．SWP では，最終的に送信側がデータの送信を（確認応答の受信も含めて）完了し，受信側が送信されたデータの受信を完了しなければならない．これは，送信完了を表す命題 all_data_acked と，受信完了を表す命題 all_data_recvd を用いて，

```
<>[](all_data_acked && all_data_recvd)
```

と記述できる．ここで，命題 all_data_acked と all_data_recvd には次の定義を用いる．

```
#define all_data_acked \
        sender[p_snd]:sbuf == 0 && \
        sender[p_snd]:remains == 0
#define all_data_recvd \
        receiver[p_rcv]:rnxt == DATA_SIZE
```

変数 p_snd と p_rcv は，それぞれ送信側と受信側のプロセス ID を表す．単体系では，プロセス sender, receiver はどちらも一つしか生成されないため，必ずしもプロセス ID を明示的に指定する必要はない．ここでは，後述する冗長系モデルにおける LTL 式の記述を簡略化するために導入している．

SWP の単体系モデルが，この LTL 式を満たすことは，SPIN によるモデル検査で確認できる（実際には，無限にメッセージの損失が発生するループや，受信ウィンドウ rwnd が常に 0 になるループを排除する必要がある）．

モデル検査により仕様の確認を行った単体系モデルと LTL 式は，以降のステップで冗長系モデルとその検査仕様の雛形として利用する．

3.2 同期なし冗長系モデルの作成

冗長系モデルは，単体系モデルに対して，(1) 対向・冗長系のプロセス宣言を修正し，(2) 現用故障の記述を追加し，(3) 送信処理の振分け処理を追加することで作成する．ここで作成する冗長系モデルは，反例の抽出を目的としており，同期機構は導入しない．

付録にある SWP 単体系モデルを雛形に送信側を二重化する場合，まず，プロセスの宣言を以下のように変更する．

```
proctype sender(chan in,out;bool act) {...}
proctype receiver(chan in,out_a,out_s) {...}
```

冗長系（sender）の引数 act は，現用であれば true となる変数であり，現用と予備で動作を変更するために用いる．対向（receiver）の引数（out_a, out_s）は，それぞれ現用と予備に対する送信用チャネルである．

次に，現用の故障を表現する大域変数 failed を定義する．下記のように，現用はイベント待ちにおける任意のタイミングで failed の値を true に設定し，処理を停止することで故障を表現する．

```
do /* event loop */
   ...
:: atomic {
   act
-> failed = true;
   false
}
od
```

更に，現用の故障時は，対向からのメッセージが予備だけに届き，予備からのメッセージが対向に届くよう制御する必要がある．この処理は，下記のように failed 及び act の値に応じて処理を振り分けることにより実現できる(注1)．

(注1)：変数 t は，パケットの損失をカウントする 1 ビットカウンタであり，パケットが無限に損失する実行系列を検査対象から排除するために用いる．

NII-Electronic Library Service

The Institute of Electronics, Information and Communication Engineers


```
inline send_msg(s,l) {
  if
  ::  (act ^ failed) && nfull(out)
  -> out!msg(s,l)
  ::  (act ^ failed)
  -> t = 1 - t
  :: !(act ^ failed)
  -> skip
  fi
}
inline send_ack(a,w) {
  if
  :: !failed && nfull(out_a)
              && nfull(out_s)
  -> out_a!ack(a,w);
     out_s!ack(a,w)
  :: !failed && nfull(out_a)
  -> out_a!ack(a,w);
     t = 1 - t
  :: !failed && nfull(out_s)
  -> out_s!ack(a,w);
     t = 1 - t
  :: !failed
  -> t = 1 - t
  ::  failed && nfull(out_s)
  -> out_s!ack(a,w)
  ::  failed
  -> t = 1 - t
  fi
}
```

また，対向には現用と予備に同じメッセージを届けるため，両方に対してメッセージを送信するよう変更を加える．ここでは，対向に対して修正を加えているが，これは現用と予備が接続するリンクの備えるブロードキャスト機能をモデルに導入するためである．よって対向の本質的な動作は変更されない．

この修正を加えたプロセス sender 及び receiver の起動は，次のように行う．ここでは，**3.1** で述べたプロセス ID を取得しており，送信側のプロセス ID (p_snd) として，現用故障後の動作に着目するため，現用ではなく予備のプロセス ID を利用する．

```
chan rs_a = [CHAN_SIZE]of{mtype,byte,byte};
chan rs_s = [CHAN_SIZE]of{mtype,byte,byte};
chan sr   = [CHAN_SIZE]of{mtype,byte,byte};
init {
  atomic {
    p_act = run sender(rs_a,sr,true);
    p_snd = run sender(rs_s,sr,false);
    p_rcv = run receiver(sr,rs_a,rs_b);
  }
}
```

以上の変更を単体系モデルに加えることにより，単体系モデルから同期機構を含まない冗長系モデルを作成できる．

3.3 モデル検査による反例の抽出

作成した冗長系モデルに対して，モデル検査を実施する．SPIN によるモデル検査では，モデルが期待した性質を満たさない場合に，検査項目に違反する実行系列が反例として出力される．そのため，この反例を同期機構の検討に活用できる．

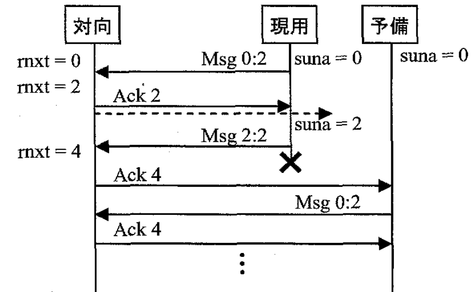


図 2　反例となる実行系列（SWP）
Fig. 2　A counterexample sequence (SWP).

冗長系モデルの検証に用いる LTL 式は，単体系モデルの検証に用いた LTL 式をもとに作成する．前述のとおり，単体系において対向–現用の振舞いを記述したものを，対向–予備の振舞いとして記述し直す必要がある．これは，プロセス ID を利用して LTL 式を記述することで対応できる．SWP の場合，単体系モデルでは変数 p_snd の値が現用のプロセス ID であり，これを利用して LTL 式を記述している．一方，冗長系モデルでは，p_snd の値は予備のプロセス ID となる．これにより，同じ LTL 式を用いて対向–予備間の仕様を表せる．

ただし，冗長系では単体系に比べて並列に動作するプロセス数が増加するため，本質的ではない実行系列が検査の対象に含まれることがある．そのような実行系列は，検査対象から除外したい．これを実現するには，必要に応じて，単体系モデルの LTL 式の一部を修正する．例えば，フェイルオーバ時の動作確認という観点からは，現用に故障が発生しない実行系列は本質的ではないため，検査対象から除外したい．これは，現用が故障しないことを表す LTL 式（`[]!failed`）と単体系モデルにおける LTL 式の選言をとることによって実現できる．現用に故障が発生しない実行系列は LTL 式 `[]!failed` を満たすため，検査式全体に違反することはない．結果として，検査対象から除外される．

以上で述べた同期機構を含まない冗長化を施したモデルに対し，二重化対応済みの LTL 式を用いてモデル検査を行った結果，**3.2** で作成した同期なし冗長系 SWP モデルでは，例えば反例として図 2 に示す実行系列が出力される（この反例の分析は次節で行う）．

3.4 反例の分析と同期機構の導入

モデル検査によって反例が抽出された場合は，その

NII-Electronic Library Service

The Institute of Electronics, Information and Communication Engineers

反例を分析して必要な同期機構を冗長系モデルに導入する．同期機構の目的は，イベントの前後における現用と予備の致命的な状態遷移の不一致が，実行系列のどの時点で生じるかを分析し，その原因となる内部状態の不一致を抑制することにある．

外部からのイベントを制御することで二重化を実現する場合，同期処理は送受信におけるタイミングの制御やメッセージの書換え処理に限定される．ただし，実際にターゲットとする実装（プロトコルスタック）がある場合は，スタック外部から参照可能なパラメータも同期機構で利用できる．

冗長系モデルの検証では，最終的に反例が得られなくなるまで，モデル検査による反例の抽出と同期機構の修正を反復する．

図 2 で示した同期なし冗長系 SWP モデルにおける反例では，予備が送信していない Msg 2:2 を対向が受信したタイミングで現用が故障している．図中，破線は，ネットワーク内でのパケットロスやネットワークインタフェースでのバッファあふれなどにより，パケットが相手に届かないことを表している．具体的には，Ack 2 は現用には届いたがパケットが損失したために予備には届かない状態で，その後の処理が進んでいる． 現用は Ack 2 を受け取った結果，確認応答未受信の最小シーケンス番号 suna を 2 に更新し，次のメッセージ Msg 2:2 を送信するが，その直後に故障している．故障の時点で，suna の値は現用の方が大きく，現用の内部状態が予備に対して先行した状況になっている．その後，対向はメッセージ Msg 2:2 を受理して Ack 4 を応答するが，予備側では Ack 4 が未送信のデータに対する確認応答であるため，Ack 4 を破棄して Msg 0:2 の再送を行う．以降，対向–予備間では，この処理が繰り返され，データの送受信が正常に行われない．つまり，内部状態 suna について現用が予備に先行した結果，現用と予備の状態遷移が一致しないことが分かる．また，他の反例を分析することで，内部状態 suna と同様に，送信の最大シーケンス番号 smax についても，現用が予備に先行した場合に致命的な状態遷移の不一致を生じさせることが分かる．

このような不一致に対処するため，現用・予備それぞれの ACK 受信時に次のような処理を追加する．suna の不一致を抑制するため，現用が予備では未送信のデータに対する確認応答を受け取った場合，予備に対して有効な ACK 番号に修正して受信する．これにより，現用では，予備に対して有効な ACK 番号だけが届くようになるため，上述の状況は抑制できる．また，smax の不一致を抑制するため，予備では，ACK 番号が送信済みのシーケンス番号より大きい場合でも，現用の smax 以下の範囲であれば，ACK 番号を予備の smax に修正して受信する．この結果，現用が送信済みかつ予備が未受信のデータに対する確認応答が届いた場合に，予備側が現用に追い付けるようになる．これらを合わせて，以下の同期機構を導入する．

```
recv_ack(_ack,_wnd);
if
::  act && _ack > sender[p_snd]:suna
-> _ack = sender[p_snd]:suna
:: !act && smax < _ack
        && _ack <= sender[p_act]:smax
-> _ack = smax
:: else
-> skip
fi;
```

この同期機構により，現用と予備における状態の不一致を同期機構が吸収できるため，対向と予備は，現用が故障しても通信を継続できるようになる．同期機構の本質的な動作は，現用と予備における内部状態の不一致が致命的な不具合を生じさせないよう，一方が他方の処理を待つ，または追い付くようなイベント制御を行うことである．上述の suna についての処理では，現用が受信した確認応答を予備の受信が確認できるまで遅延させるなど他の方式を用いても不一致を回避できるが，ここでは，処理が簡潔な上記の方式を採用した．

最終的に，モデル検査によって反例が得られないことを確認し，検証は完了となる．

4. TCP への適用

SWP モデルは，主にデータ通信を行うプロトコルをモデル化しているが，実際の通信プロトコルには，セッションの確立や切断など様々な手続きが含まれる．そこで，提案手法の有効性を確認するため，TCP の二重化を例に提案手法の適用を試みた．

4.1 プロトコル概要とモデル化

TCP は RFC 793 [10] により，基本的な技術仕様が策定されているデータ転送プロトコルである．TCP には，ふくそう制御や Selective ACK など多くの改良が加えられている．本論文では，モデルを簡略化するためにこれらの拡張は取り入れず，RFC 793 に従いモデルを作成した．

TCP を用いた通信の処理は，

- コネクション確立

NII-Electronic Library Service

The Institute of Electronics, Information and Communication Engineers

- データ通信
- コネクション切断

の三つに分類され，この分類に従いモデルを作成して検査を実施する．モデルの作成に際し，状態数の増加及び検査式の複雑化を防ぐため，以下の制限を加える．

まず，データの送信方向を片方向に限定する．双方向のデータ送受信を許すと，状態数は片方向の場合に比べて2乗程度になるおそれがあるため，送信側と受信側の二つに分けて検査を行う．片方向の通信とすることで，モデル検査によって抽出できない反例が生じる可能性がある．しかし，TCPの上位レイヤにあるプロトコルがリクエスト/レスポンス形式であれば，ある時点における通信はその大半が片方向であると想定されるため，この制限を加えても多くの場合をカバーできると考えられる．

次に，2MSL タイマを無効化する．つまり，TIME_WAIT から CLOSED への状態遷移は発生させない．2MSL タイマによる TIME_WAIT から CLOSED への遷移があると，TIME_WAIT から異常系のシーケンスで CLOSED に遷移した場合を考慮する必要が生じるため，検査式の記述が複雑になり，状態数が増加してしまう．TIME_WAIT 状態は，実質的にはコネクションの切断完了とみなせるため，2MSL タイマによる遷移を無効化することで，検査式を単純化できる．

最後に，再送タイムアウトの制限をなくす．2MSL タイマの場合と同様，最大再送回数を超えた場合の異常系処理を記述すると，モデルと検査式が複雑になる．最大再送回数に達するまでの時間は十分長いと考えられるので，ここでは無限に再送されるものとしてモデル化する．

なお，モデルに利用する各種パラメータとして，チャネルのバッファ長2，送受信するデータの長さ4，最大受信ウィンドウサイズ3，最大セグメントサイズ2を用いる．

4.2 検査項目

TCPにおける検査項目は，前述した確立・通信・切断のそれぞれについて個別に定義する．

コネクションの確立では，最終的にコネクションの確立が完了する必要がある．これを検査するには，通信している両ノードがともに ESTABLISHED 状態になることを確認すればよい．両者がともに ESTABLISHED 状態であることを established で表せば，

```
<>[]established
```

なる LTL 式により，セッションが正常に確立されることを検査できる．

同様にデータ通信では，データの送信が完了する必要がある．データの送信は片方向に制限しているため，SWP と同じ LTL 式

```
<>[](all_data_acked && all_data_recvd)
```

を検査式として利用できる．

コネクションの切断では，コネクションが正常に切断される必要がある．本来ならばコネクションの切断後は，最終的に両者共 CLOSED 状態になる．しかし，上述のとおり 2MSL タイマを無効化しているため，TIME_WAIT 状態も正常な切断状態とみなす必要がある．両ノードの状態が CLOSED，TIME_WAIT のいずれかであることを closed_or_timewait で表すと，コネクションの切断が正しく行われることは，

```
<>[]closed_or_timewait
```

によって検査できる．

4.3 反例の抽出

提案手法の手順に従い TCP の冗長化モデルを作成し，前述の検査項目についてのモデル検査により，反例を抽出した．コネクションの確立及び切断において抽出された反例のシーケンスを図3に示す．これら以外に，データ通信における検査によって得られた反例もあるが，図2で示した SWP の反例と同様であるため，省略する．

(a)，(b) はそれぞれ，クラスタ側がパッシブオープン，アクティブオープンを実行する場合のシーケンスである．(a) では，予備が SYN を受信する前に，対向が ESTABLISHED 状態に移行している．現用の故障によって，対向–予備間の通信が開始されると，予備は RST パケットでコネクションを強制的に切断する．(b) では，予備が SYN パケットに対する SYN-ACK を受信できないまま，対向–現用間でコネクションが確立し，その後に現用が故障している．対向では，既にコネクションが確立しているため，SYN-ACK は送信されない．そのため，予備ではその後の通信を行うことはできない．

(c) は，コネクションの切断における反例の一例である．(c) では，対向と予備がほぼ同じタイミングでアクティブクローズを開始し，FIN_WAIT_1 状態に移行している．現用は，予備とほぼ同じタイミングでアクティブクローズを開始しようとするが，ここでは予

NII-Electronic Library Service

The Institute of Electronics, Information and Communication Engineers

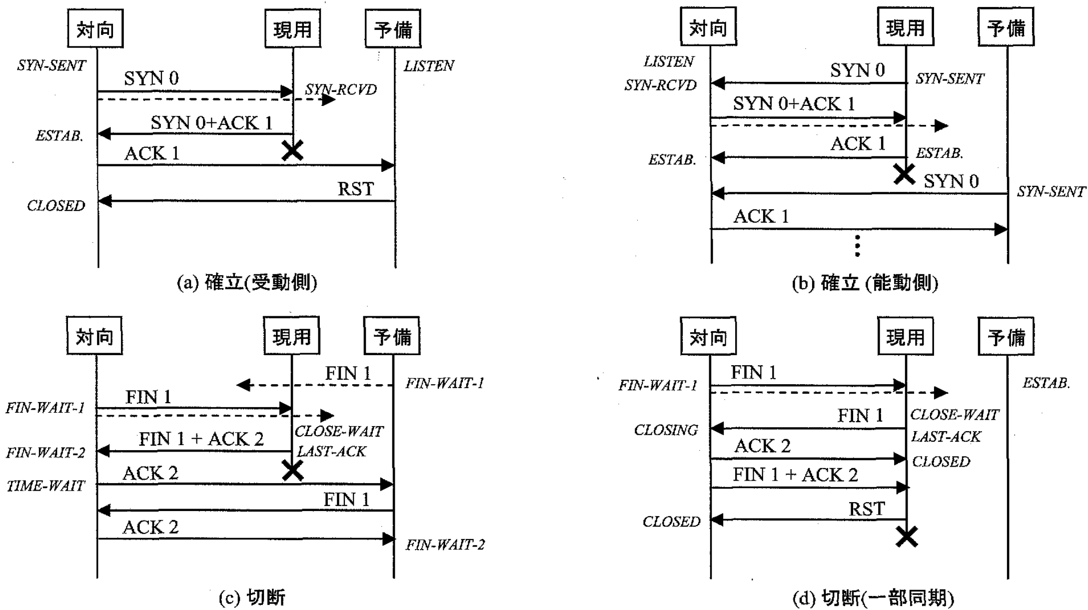


図 3　反例（TCP）
Fig. 3　Counterexample sequences (TCP).

備よりわずかに動作が遅れた状態で，対向からのFINが先に届いてしまい，結果的にパッシブクローズの動作になっている．予備からのFINは，現用が稼動中なので対向には送信されない．対向からのFINは現用には届いているが，予備には届いておらず，加えて，現用はパッシブクローズ側の状態遷移に移っているため，現用と予備の状態に不一致が生じている．対向は，FINパケットに対し，現用からACKとFINを受信してTIME_WAIT状態に移行する．一方，予備は自らが送信したFINパケットに対するACKパケットは返ってくるが，対向からのFINパケットが届いていない．結果として，予備はFIN_WAIT_2状態にとどまることになる．この状態では，上位レイヤのアプリケーションに対してコネクションの切断が通知されない．

4.4　同期機構の導入

データ通信における送信側二重化時の同期処理は，現用の先行動作をある程度許容し，予備が未送信のデータに対してACKを受けた場合は，ACK番号を調整することで先行の状態に追い付けるよう修正した．また，受信側の二重化については，SWPの同期機構を流用できる．

コネクションの確立では，予備がコネクションを確立できないまま，対向–現用間でコネクションが確立してしまう可能性がある．現用の先行動作を許さないためには，現用と予備が受信したパケットのSYNビットを調べ，ESTABLISHED状態に至るまでの状態遷移を逐次管理する方法が考えられる．しかし実際には，SYNがシーケンス番号を消費するため，データ通信用に導入した処理が有効に働くことから，この状態遷移の逐次管理は不要となる．

これに対し，コネクションの切断においては，データ通信用に導入した同期だけでは不十分である．FINパケットはSYNパケットと同様にシーケンス番号を消費するため，一見FINにおいてもデータ通信の同期機構が有効だと予想されるが，データ通信用の同期のみを導入した場合は反例(d)が得られる．

本来，CLOSE_WAITまたはLAST_ACKで送信されるACK番号はFINに対するものでなければならない．しかし，現用から送信されるACK番号は，同期機構のために予備のACK番号で書き換えられる．つまり，現用はFINに対する確認応答を送ったつもりでも，対向には届いていない状態が生じる．この影響で，アクティブクローズを実行した対向がTIME_WAITに移行しないうちに，現用がCLOSED状態に至る．結果として，対向からはコネクションがRSTパケットによって強制的に切断されたように見える．

これを改善するために，FINの同期機構を導入した．具体的には，予備がFINを受信していない状態で現用がFINを受信した場合は，FINビットをクリアする．

4.5　同期機構の検証

最終的に，**4.3**で挙げた検査項目に対する反例が検出されなくなるまで，反例の抽出とモデル（同期機構）

NII-Electronic Library Service

The Institute of Electronics, Information and Communication Engineers

表 1 検査結果（TCP）
Table 1 The result of model checking (TCP).

	状態数	hash factor	時間 (s)
確立（能動）	1.6×10^5	1.8×10^5	6.1
確立（受動）	8.3×10^5	3.9×10^4	39.
通信（受信）	1.0×10^{10}	3.2	4.5×10^5
通信（送信）	1.6×10^{10}	2.1	6.9×10^5
切断	2.6×10^8	1.3×10^2	1.3×10^2

の修正を繰り返した．TCP の二重化モデルは Promela で 1,000 行程度のモデルとなった．作成したモデルは状態数が多く，利用した計算機環境では全数検査ができないため，近似を利用して検査を実施した．モデル検査は，bitstate hashing [11] による近似を有効にし，ハッシュテーブルのサイズとして 4 GByte を指定した．モデル検査には，Linux 2.6/Xeon 3.8 GHz/5 GByte の汎用機を利用した．表 1 に検査結果を示す．

hash factor は，bitstate hashing を利用した場合の探索空間に対するカバー率の指標である [12]．hash factor (Hf) は，到達した状態数を N，利用メモリのビット数を M とすると $Hf = \frac{M}{N}$ で定義される．hash factor が 10 以上のとき，平均で 98%以上のカバー率を期待できる．

到達状態数は，データ通信の送信側を二重化する場合が最も多く，1.6×10^{10} 個に及んだ．このとき，検査完了までに約 8 日を要した．

5. 考　察

TCP の二重化に対して提案手法を適用した結果，データ通信だけでなくセッションの確立・切断シナリオにおいても，同期機構が不十分な場合に反例を抽出できた．これら三つの機能は多くの通信プロトコルが備えることから，本論文で取り上げた例以外の様々な通信プロトコルにも本手法は適用可能であると考えられる．また，反例によって各ノードの動作を完全に再現できるため，テストの結果をトレースする場合に比べて，不具合の原因を容易に特定できる．

一方で，より複雑なプロトコルを扱うには課題が残る．二重化によりモデル上で並列に動作するプロセスが増えるため，探索する状態数が増加し，結果として検証が困難になる．探索空間に対するカバー率の観点からは，今回の TCP 二重化における hash factor の最低値 2.1 という値は必ずしも十分ではない．しかし，hash factor は利用メモリ量を増やすことで改善されるため，その値を 10 以上に改善することは十分可能である．また，複数回の検査を行う sequential bitstate hasing [12] を用いてもカバー率は改善できる．

本論文では，具体例として TCP の二重化検証を取り上げたが，提案手法は汎用性があり，プロトコルの単体系モデルとその検査式を定義できる，その他の通信プロトコルに対しても有効であると考えられる．その場合，**3. 2** に示した手順によって，同期機構なしの二重化モデルとその検査式が機械的に生成でき，**3. 3** の手順で，二重化モデルの検査が可能となる．本論文では取り上げなかったが，筆者らは，データリンク層の通信プロトコルである Alternating Bit Protocol [13] でも，提案手法の有効性を確認している．

6. む す び

本論文では，モデル検査を利用した通信プロトコル二重化の検証手法を提案した．提案手法により，通信プロトコルの二重化に対して，単体系における検証資源を活用した設計段階での検証が可能となる．また，TCP 二重化に対して提案手法を適用することで，設計段階での不具合を検出可能であることを確認し，提案手法の有効性を示した．

文　献

[1] P. Weygant, Clusters for High Availability: A Primer of HP Solution, Prentice Hall, 2001.

[2] “HomePage:Linux HA,” http://www.linux-ha.org/

[3] Z. Shao, H. Jin, B. Chen, J. Xu, and J. Yue, “HARTs: High availability cluster architecture with redundant TCP stacks,” International Performance Computing and Communication Conference (IPCCC), pp.253–260, 2003.

[4] 狩野秀一，地引昌弘，“トランスポート端点のポータブルなクラスタ方式，” 信学技報，IA2007-2, May 2007.

[5] D. Chkliaev, J. Hooman, and E.D. Vink, “Verification and improvement of the sliding window protocol,” Tools and Algorithms for the Construction and Analysis of Systems, vol.2629, pp.113–127, LNCS, 2003.

[6] P.R. D'Argenio, J.P. Katoen, T. Ruys, and G.J. Tretmans, “The bounded retransmission protocol must be on time!,” Tools and Algorithms for the Construction and Analysis of Systems, vol.1217, pp.416–431, 1997.

[7] A. Endres and D. Rombach, A Handbook of Software and Systems Engineering: Empirical Observations, Laws and Theories, Addison Wesley, 2003.

[8] E.M. Clarke, O. Grumberg, and D. Peled, Model Checking, The MIT Press, 1999.

[9] G.J. Holzmann, The Spin Model Checker: Primer

NII-Electronic Library Service

The Institute of Electronics, Information and Communication Engineers

and Reference Manual, Addison Wesley, 2003.
[10] J. Postel, "RFC793: Transmission control protocol," Sept. 1981.
[11] G.J. Holzmann, Design and Validation of Computer Protocols, chapter 11, Prentice Hall, 1991.
[12] G.J. Holzmann, "An analysis of bitstate hashing," Formal Methods in System Design, vol.13, no.3, pp.289–307, 1998.
[13] K.A. Bartlett, R.A. Scantlebury, and P.T. Wilkinson, "A note on reliable full-duplex transmission over half-duplex links," Commun. ACM, vol.12, no.5, pp.260–261, 1969.

付　　録

SWP 単体系モデル

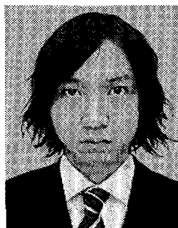
```
#define MAX_RWND        3
#define MSS             2
#define CHAN_SIZE       2
#define DATA_SIZE       4
mtype = { msg, ack };
bit t;
pid p_snd, p_rcv;
inline send_msg(s,l) {
  if
  :: nfull(out)
  -> out ! msg(s,l)
  :: true
  -> t = 1 - t
  fi;
  s = s + 1;
  smax = (smax > s -> smax : s)
}
inline send_ack(a,w) {
  if
  :: nfull(out)
  -> out ! ack(a,w)
  :: true
  -> t = 1 - t
  fi
}
inline recv_msg(s,l) { in ? msg(s,l) }
inline recv_ack(a,w) { in ? ack(a,w) }

proctype sender(chan in, out)
{
  byte swnd, snxt, suna, sbuf, smax;
  byte _ack, _wnd, _len;
  byte remains = DATA_SIZE;
  do /* event loop */
  :: atomic {
     recv_ack(_ack,_wnd);
  -> if
     :: suna < _ack && _ack <= smax
     -> sbuf = sbuf - (_ack - suna);
        suna = _ack; swnd = _wnd
     :: else
     -> skip
     fi;
     _ack = 0; _wnd = 0
  }
  :: atomic {
     swnd - snxt + suna > 0 &&
     sbuf - snxt + suna > 0
  -> _len = (MSS < sbuf - snxt + suna ->
              MSS : sbuf - snxt + suna);
     send_msg(snxt,_len);
     _len = 0
  }
  :: atomic {
     !swnd && snxt == suna && sbuf
  -> send_msg(snxt, 1)
  }
  :: atomic {
     snxt > suna
  -> snxt = suna
  }
  :: atomic {
     remains
  -> sbuf++; remains--
  }
  :: atomic {
    !remains && !sbuf
  -> break
  }
  od
}

proctype receiver(chan in, out)
{
  byte rwnd, rnxt, _seq, _len;
  do
  :: atomic {
     recv_msg(_seq,_len);
  -> if
     :: rnxt <  _seq || rnxt >= _seq + _len
     -> skip
     :: else
     -> _len = _seq + _len - rnxt;
        _len = (rwnd<_len -> rwnd:_len);
        rwnd = rwnd - _len;
        rnxt = rnxt + _len
     fi;
     send_ack(rnxt, rwnd);
     _seq = 0; _len = 0
  }
  :: atomic {
     rwnd < MAX_RWND
  -> rwnd++
  }
  od
}

chan rs = [CHAN_SIZE] of {mtype,byte,byte};
chan sr = [CHAN_SIZE] of {mtype,byte,byte};
init {
  atomic {
    p_snd = run sender   (rs, sr);
    p_rcv = run receiver (sr, rs);
  }
}
```

（平成 21 年 9 月 28 日受付，22 年 1 月 22 日再受付）



池田　聡　（正員）

2007 京都大学大学院情報学研究科知能情報学専攻修士課程了．同年，NEC 入社．現在，同社システムプラットフォーム研究所にてプロトコル・エンジニアリング技術の研究開発に従事．

NII-Electronic Library Service

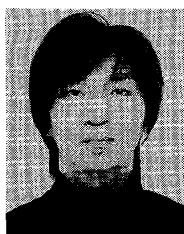
The Institute of Electronics, Information and Communication Engineers

地引　昌弘　（正員）

平 4 東工大大学院修士課程了．同年，NEC 入社．平 15 筑波大大学院博士課程了．現在，NEC 中央研究所に所属．博士（システムズ・マネジメント）．平 18 より和歌山大システム工学部客員教授を兼任．平 21 より本会英文論文誌 B 編集委員．ネットワーク制御，分散システム，ソフトウェア科学などに興味をもつ．

久野　　靖

1984 東京工業大学理工学研究科情報科学専攻博士後期課程単位取得退学．東京工業大学助手，筑波大学講師・助教授を経て現在，筑波大学ビジネス科学研究科教授．プログラミング言語，ユーザインタフェース，情報教育などに興味をもつ．著書に，「UNIX の基礎概念」,「Ruby による情報科学入門」などがある．



西森　丈俊

1984 東京理科大・理工・数学卒．同年ナムコ（株）入社．家庭用ゲーム開発に従事．1996 よりドリームファクトリー（株）入社．2001 よりフリーランス．

NII-Electronic Library Service