Coqを使ったツリー型ネットワークトポロジ上でのCCNのモデル化と検証について

森嶋 崇[1]　後藤 瑞貴[1]　森口 草介[1]　高橋 和子[1]

概要：Content-Centric Networking(CCN) とは 2009 年に Van Jacobsen が提案した通信方式であり，アドレスを利用するのではなくコンテンツ名に注目して通信をおこなうものである．CCN では中継ノードでコンテンツをキャッシュすることができ，ネットワークの利用効率の向上や，応答時間の短縮が特徴として挙げられる．現在はシミュレーションをベースとして動作や性能のチェックがおこなわれているが，CCN は確立した技術ではないため，実用化にむけて動作の正当性の検証が望まれる．本発表では，証明支援系 Coq を用いて，CCN のプロトコルを帰納的にモデル化し，二分木のツリー型ネットワークトポロジにおいて動作の正当性の検証をした．このモデルでは，各ノードでおこなわれているマッチング処理を実装し，1 つの時系列リストを用意して，ノード間のパケットの送受信すべてを管理するようにした．動作の正当性として，あるコンテンツがネットワーク上に存在し，ユーザがそれを要求すれば，必ず正しいものを受信できるかということと，その逆の，コンテンツを受信した場合は，そのユーザが要求を送っていたということを証明した．

Formalization and Verification of CCN Protocol on a Tree Topology Using a Proof Assistant Coq

Takashi Morishima[1]　Mizuki Goto[1]　Sosuke Moriguchi[1]　Kazuko Takahashi[1]

Abstract: Content-Centric Networking (CCN) is a communication architecture which was developed by Van Jacobsen on 2009. Communication on CCN is based on the names of objects, rather than on addresses. CCN can store contents on relay nodes. It is said that CCN improves network efficiency and reduces response time, but it is not a well-established technique. Although its behavior and performance are explored mainly by simulation, the behavioral correctness is required to be verified for its practical use. In this model, the matching process undertaken on each node is implemented and all events of sending/receiving packets are managed by a unique list. We proved two properties as behavioral correctness of this model on a binary tree topology. First, a user can retrieve the content if it sent the request under the condition that the content exists in the network. Second, a user must have been sent the request for the content if a user receives it under the condition that the content exists in the network. We have implemented the model and proved these properties using a proof assistant Coq.

1. 序論

インターネットでパケット通信の構造ができて以来，約 50 年で既に 500 エクサバイトものコンテンツがネットワーク上に存在している．しかし，ユーザはコンテンツがどこから得られるかということには関心がなく，コンテンツを得ることができれば十分である．これは現在の，“どこ”という“アドレス”を利用したインターネットの構造と矛盾していることになる．本研究の対象となる Content Centric Networking (以下 CCN)[5] は“なに”という“コンテンツ名”に注目した新しい通信方式である．従来の通信では，ア

[1] 関西学院大学
Kwansei Gakuin University
理工学部情報科学科高橋和子研究室〒 669–1337 兵庫県三田市学園 2–1 Tel: 079-565-8391

ドレスを利用して通信をしていたが，CCN はオブジェクトの名前で通信をおこなう．CCN の利点として，コンテンツを任意の中継ノードでキャッシュできるので，ネットワークの利用効率の向上や，応答時間の短縮が挙げられる．ただし，CCN は現在も性能評価がおこなわれており，確立された技術ではなく，動作の正当性の研究はされていない．

正当性は，様々な保証を定式化したものの複合体として示される．この保証の選び方には決まりがなく，検証する目的に応じて選択する必要がある．ここでは，正当性とは各入力に対して正しい出力を生成することとする．これを通信に関して当てはめると，あるノードが要求を出せば，正しい応答が返ってくるということである．このような正当性を検証した事例 [2][12] はあるが，CCN を対象とした研究はされていない．

著者らはこれまでに，CCN のプロトコルを証明支援系 Coq で帰納的にモデル化し，正当性として大きく 2 つの検証項目を証明した [13]．ただし，その研究ではノード数と各ノードの接続を固定したモデルのみを対象としていた．

本研究では，固定トポロジではなく，ノード数を任意とした完全二分木のネットワークトポロジにおいてを対象として動作の検証をする．Coq を使用することで，ある計算プログラムが誤っているかどうか机上の証明では簡単に保証できないようなものに，厳密な証明を与えることができる．具体的には，正当性の 1 つとして，ユーザがあるコンテンツを要求すれば，必ず正しいものを受信できることを検証した．本発表の構成は，以下の通りである．

第 2 章では形式的手法について述べる．第 3 章では CCN の概要を述べる．第 4 章では CCN のモデル化について述べる．第 5 章では Coq での検証について述べる．第 6 章では今回提示したモデルの有効範囲について述べる．第 7 章ではまとめと今後の課題について述べる．

2. 形式的手法

一般に対象をプログラムで扱えるようにモデル化し，要求される仕様を満たしているか機械的に計算し検証する方法を形式的手法という．形式的に問題を捉えることでさらに厳密な安全性の検証や複雑な環境での検証が可能になる．ただし，どのように形式化するか，対象をどこまで深く検証するか，といった点でアプローチの手法が変わってくる．

代表的な手法としてモデル検査と定理証明の 2 つがある．モデル検査は代表的なツールに，Spin[3] や SMV[8] などがあり，起こりうる全ての状態変化を網羅的に検証する手法で，ほとんどのツールでは扱うことのできるモデルは有限である．また，あるデータに対して，ある動作をしていたときに，意図に反する動作がみられるという反例の発見を目的とするならば，モデル検査を利用すると都合が良い．変数のとる値が無限個の場合を扱えるモデル検査器も存在する [7] が，ほとんどのモデル検査器は検証を自動でおこなうため，仮にモデル化を間違えていて検証が成功しても，モデルの間違いに気付きにくい．

一方，定理証明は，ある公理系とその推論規則を使用し，要求される性質を満たしているかどうか検証する方法である．定理証明の代表的なツールに，Isabelle/HOL[9], Coq[1], PVS[11], ACL2[6], Agda[10] などがある．これらは証明支援系とも呼ばれる．定理証明は無限状態を扱えるため，モデル検査よりも検証範囲が広い．さらに，対話型の定理証明器は段階的に証明が進むため，途中で間違いに気付きやすい．

本研究では，通信方式の検証をおこなう．コンテンツ名やノード数は無限であるので，無限個の値を扱える定理証明の方が向いている．また，受信パートによって，時間的に動作が変わるネットワークにおいて，送受信イベントがうまくいくかどうかの判定を段階的におこなえる対話型の検査器が向いていると考えられる．さらに，CCN のソースコードが Java で書かれているため，後の応用を考え，他言語との互換性の多い Coq を使うことにした．

3. CCN の概要

CCN[5] は Van Jacobsen が提案した，コンテンツの名前に基づいて通信をおこなうネットワークアーキテクチャである．CCN は物理的なネットワークの構造に変更を加えることなく，ソフトウェアとしての機能を書き換えるだけで，既存のインフラストラクチャ上に構築することができる．ノードとはユーザ，ルータ，サーバの 3 種類のことである．従来の通信では，パケットは 2 つのアドレスを利用し，2 つのノード間で対話が成り立っていた．CCN では中継ノードでコンテンツをキャッシュすることができ，効率的にコンテンツを取得できる．

TCP/IP と CCN の構造の違いは以下の 3 点である．まず，通信で使うパケットの種類は，TCP/IP では単独なのに対して，CCN では要求と応答の 2 種類に分けている．次に，TCP/IP ではパケットに行き先のアドレスを持たせていたのに対して，CCN ではコンテンツ名を持たせている．また，TCP/IP ではノードはバッファの機能しか持たないが，CCN では新たに 3 種類の構成要素を持つ．

3.1 CCN のパケット

CCN は従来のようにパケットを利用した通信ではあるが，Interest パケット (要求) と Data パケット (応答) の 2 種類で構成される．Interest パケットは，Content Name と Selector，Nonce を持つが，今回は Content Name のみに着目し，コンテンツ名のみを持つとした．このパケットは，ユーザがサーバに向けてコンテンツを要求する際に用いる．Data パケットは，Content Name，Signature，Signed Info，Data の 4 つを持つが，Interest パケットと同様に，

今回は Content Name と Data の 2 つに着目し，コンテンツ名とそのコンテンツデータを持つとした．このパケットは，サーバがデータを返送する際に用いられる．

3.2 CCN の各ノードが持つ構成要素

各ノードの持つ構成要素は，従来のようにパケットをキャッシュする機能だけではなく，Forwarding Information Base(以下 FIB)，Pending Interest Table(以下 PIT)，Content Store(以下 CS) の 3 種類である．

FIB は，Interest パケットが目的のコンテンツを持つサーバにたどり着くように転送するのに用いられる (図 1)．FIB にもコンテンツ名が含まれ，Interest パケットのコンテンツ名と比較し，一致すれば目的のコンテンツを保持するサーバに近い側のノードへ送信することができる．

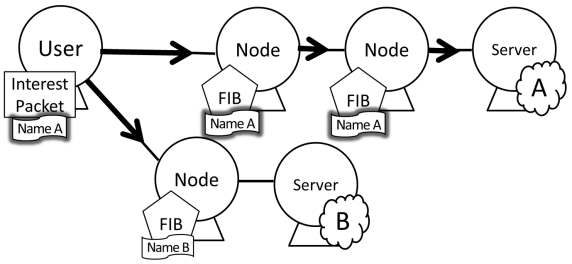

図 1　Forwarding Information Base

PIT は，ユーザからサーバに向けて Interest パケットが通過した軌跡を記録するリストで，各ノードがリストを持っている．FIB と同様に PIT にもコンテンツ名が含まれ，コンテンツ名ごとに異なるリストで管理している．これは，サーバからユーザに Data パケットを送信する際に，転送先を決定するのに用いられる (図 2)．

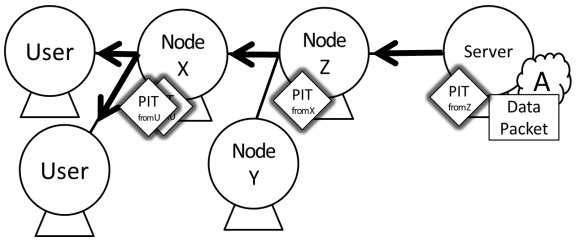

図 2　Pending Interest Table

CS はバッファで，これは従来の機能と似ているが，パケット送信後もすぐに破棄せずにできる限り Data パケットを保持し，保持しきれなくなると，利用されずに古くなったものから破棄する．この CS のおかげで，同一のコンテンツ名の要求に対して，再利用することができる．

3.3 CCN の動作

CCN では，パケットが持つコンテンツ名と構成要素が持つコンテンツ名でマッチングをおこない，どの構成要素でマッチングしたか次第でパケットの送信先が異なる．マッチングでは同時にそのノードの構成要素の内容を更新する．

3.3.1 Interest パケットに対する動作

ユーザがコンテンツ名を含む Interest パケットを，隣接する全てのノードに送信する．各ノードは Interest パケットが到着すると CS，PIT，FIB の優先度順で，パケット内のコンテンツ名と各ノード内のコンテンツ名でマッチングをおこなう (図 3)．

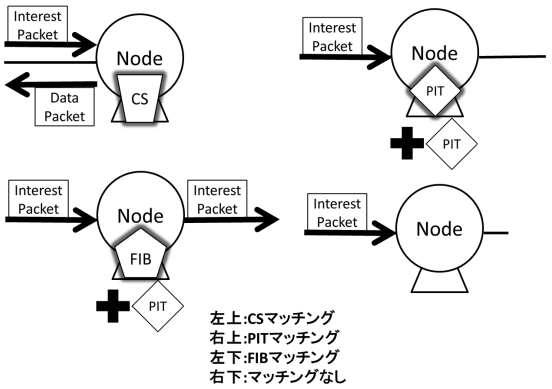

図 3　Interest パケットのマッチング

- CS でマッチングする場合

到着したノードに目的の Data パケットがあるので，Data パケットを要求のあった方に返送し，Interest パケットは破棄する．

- PIT でマッチングする場合

コンテンツ名が一致する Interest パケットが既にこのノードを通ったことがあり，FIB のマッチングが成功していれば PIT が含まれていることになるので，どこから来たかという情報 (PIT) をこのノードのリストに追加し，Interest パケットは破棄する．

- FIB でマッチングする場合

コンテンツ名が一致する Interest パケットが，初めてこのノードに到着したということなので，どこから来たかという情報 (PIT) をこのノードのリストに追加し，目的のコンテンツを保持するサーバに近い側のノードに Interest パケットを送信する．

- どれもマッチングしない場合

ノードがどの構成要素も持たず，そのコンテンツに関する情報がないので，Interest パケットを破棄する．

3.3.2 Data パケットに対する動作

Interest パケットの場合と同様に，各ノードは Data パケットが到着すると CS，PIT，FIB の優先度順で，パケット内のコンテンツ名と各ノード内のコンテンツ名でマッチングをおこなう (図 4)．Data パケットは，PIT とマッチングした場合以外はパケットが破棄されるので，PIT のマッチング以外は例外処理といえる．

- CS でマッチングする場合

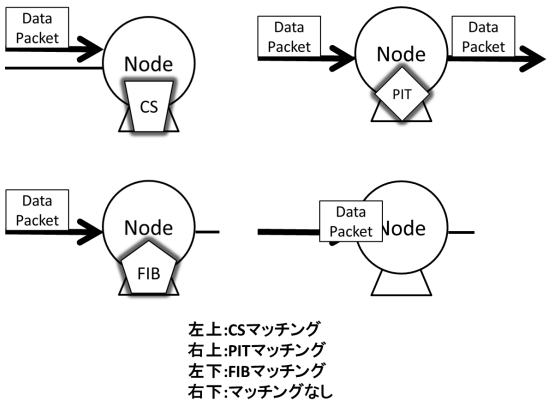

図 4　Data パケットのマッチング

既に同じコンテンツを持つ Data パケットが到着したノードに存在しているので，Data パケットを破棄する．

- PIT でマッチングする場合

到着したノードには，どこから来たかという情報がリストで残っているので，データをキャッシュしつつ (CS)，リストの情報をもとに，Data パケットを送信する．

- FIB でマッチングする場合

このノードを通った Interest パケットがなかったということになるので，Data パケットを破棄する．

- どれもマッチングしない場合

ノードがどの構成要素も持たず，そのコンテンツに関する情報がないので，Data パケットを破棄する．

3.4　CCN の動作例

動作例として図 5 のように FIB を持つモデルを考える．

まず Interest パケットの流れを説明する．ユーザは，隣接するノード A とノード B に Interest パケットを送信する．ノード A が Interest パケットを受信すると，ノード A には FIB が存在するため，FIB でマッチングをおこなう．この場合，パケットがユーザから送信されたことを記録するためにノード A の持つ PIT にこの情報が追加され，隣接するノード C に Interest パケットが送信される．ノード B が Interest パケットを受信すると，ノード B には 1 つも構成要素が存在しないため，マッチングが失敗し Interest パケットは破棄される．ノード C がノード A から Interest パケットを受信すると，ノード C には FIB が存在するため，FIB でマッチングをおこなう．この場合，パケットがノード A から送信されたことを記録するためにノード C の持つ PIT にこの情報が追加され，隣接するサーバに Interest パケットが送信される．サーバがノード C から Interest パケットを受信すると，サーバには CS と FIB が存在し，CS とのマッチングが優先されるので，サーバからノード C に Data パケットが送信され，Interest パケットは破棄される (図 5)．

次に Data パケットの流れを説明する．ノード C がサーバから Data パケットを受信すると，ノード C には PIT と FIB が存在するので，PIT で優先的にマッチングをおこなう．ノード C からユーザ側に向かってはノード A とノード B の 2 つの隣接ノードがあるが，PIT には Interest パケットを送信したのはノード A のみであるという記録があるため，ノード A のみに Data パケットが送信される．ノード A がノード C から Data パケットを受信すると，ノード A には PIT と FIB が存在するので，PIT で優先的にマッチングをおこなう．同様に PIT の情報を利用してノード A からユーザに Data パケットが送信される．この結果，Data パケットがユーザに到達し，ユーザのデータ取得が成功したことになる (図 6)．

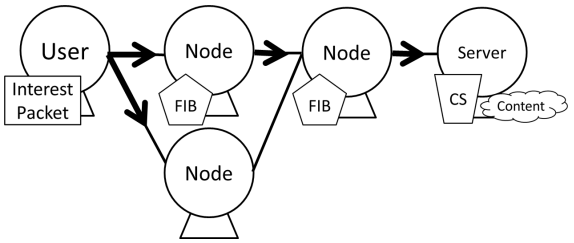

図 5　例：Interest パケットの流れ

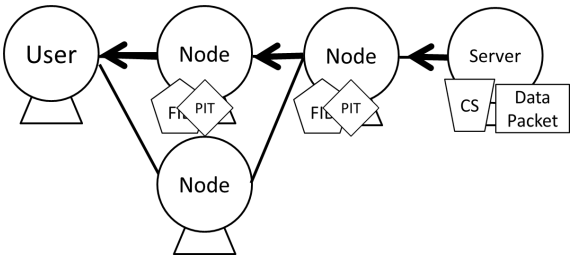

図 6　例：Data パケットの流れ

4.　CCN のモデル化

完全二分木のツリー型のネットワークトポロジを対象とし，CCN の仕組みを帰納法に基づいて Coq でモデル化する．ユーザに隣接する上流側のノードに必ず FIB が存在し，マッチングが一切起こらず通信が途絶えることはないと仮定し，オリジナルのコンテンツを，サーバに CS として持たせることで実装している．ここで，あるノードにおいて，そのノードよりもサーバに近い方を上流，遠い方を下流とそれぞれ呼ぶ．

4.1　Coq

Coq は，INRIA(フランス国立情報学自動制御研究所) で開発されている証明支援系である．Coq は型理論に基づいており，関数型言語 OCaml で書かれている．Prop(命題)，Set(項)，Type(命題と項の双方を含む) の 3 つの型を中心に，bool 型や数値型が表現されており，これらの型の制約に反する記述はできない．

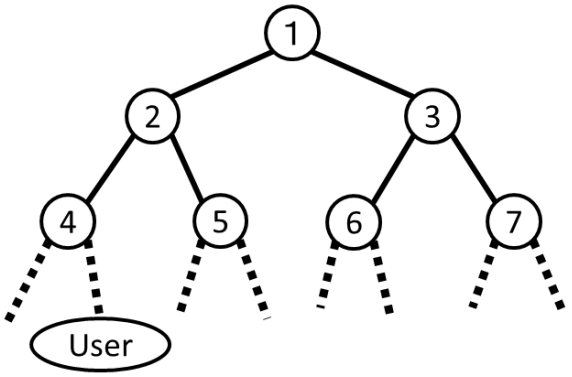

図 7　ツリー型のモデル

Coq は対話型言語の 1 つでもあり, 検証は一段階ずつ対話的におこなわれるため，証明の動作が確認しやすい．もし誤りがあった場合はそこで処理が停止するので，間違っている部分の特定がしやすい．Coq は他の証明支援系に比べ，自動で証明を進行させる機構は弱いので，証明したい補題に関しては自分でライブラリを用意する必要がある．

4.2　ネットワークトポロジ

サーバをノード 1 とする．各ノードの接続関係は，あるノード N に対して，隣接する 3 つのノードのうち，サーバに近い側を N/2，逆側を 2∗N と 2∗N+1 と表現し，木の高さに関して帰納法を適用する (図 8)．サーバを除く全ノードは Interest パケットを送信することができ，通信の開始地点となり得るとする．これは，ノードが中継地点の役割だけでなく，要求を出すことができるということである．また，サーバはネットワーク上にただ 1 つとし，少なくとも 1 つのノードが Interest パケットを送信し始めるとする．

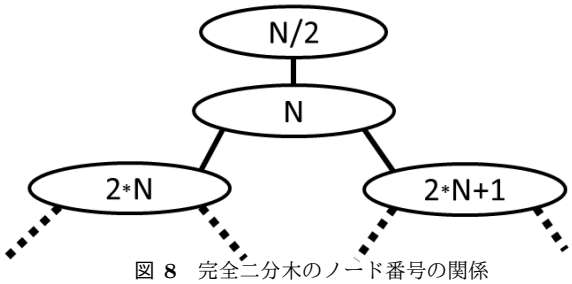

図 8　完全二分木のノード番号の関係

Coq でモデル化する上で特徴となるイベントと，CCN の特徴であるマッチングとプロトコルについて説明する．

4.3　イベント

CCN は，マッチングが起こると各ノードが持つ構成要素のうち，PIT と CS が変化する場合がある．そこで，あるノードが送信するというイベント (Say)，ノードに構成要素を追加すること (AddStruct) を Event として以下のようにそれぞれ定義する．

```
Inductive Event : Type :=
 |Say : Node → Packet → Event
 |AddStruct : Node → Struct → Event
```

“Inductive”は，帰納的な定義で用い，“| コンストラクタ名 : 型 → 型”と書く．Event は Type 型で Say と AddStruct の 2 種類のコンストラクタを持つものとして定義した．つまり，Say と AddStruct は Event 型となる．この“Inductive”の定義が Coq に受け付けられたとき，「ある命題 P が，Event のコンストラクタすべてに成り立つならば，任意の Event について P が成り立つ」といった定理が自動的に加わる．

これらのイベントを 1 本の時系列リストで管理し，現在のイベントリストを調べ，対象となるイベントを見つけてそこから構成要素の情報を抽出する．図 9 に，図 5 のイベントをリストで示す．イベントが新しく起こるたびに，リストの先頭に追加する．例えば，図 9 の Say U Interest は，ユーザが Interest パケットを送信したイベントを表し，Add PIT A from U は，ノード A に，ユーザから来たという PIT を追加するイベントを表す．

これにより，各ノードの持つ構成要素が様々に変化し状態数が膨らむ場合であってもモデル化することができる．

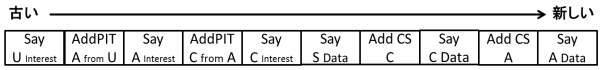

図 9　動作例 (図 5) のイベントをリストで示したもの

同じノードがパケットの送信をやり直すときなどに，同じイベントが何度も起こる場合がある．これは，ユーザが何度も同じ要求をする DoS 攻撃に近い形である．同一イベントを重複してリストに追加していくと，CCN のプロトコルにしたがってパケットが上流に向かうといえない．これを回避するために，本研究では，同じイベントは 1 度だけ認めて 2 度目以降はイベントリストに追加しない規則を，再帰的に定義した関数 list_check でそれが真か偽かを表現し, 対応した．なお，この DoS 攻撃を低減させる別の手法も提案されている [4].

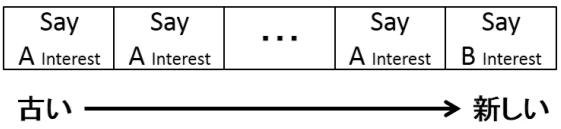

図 10　同じイベントばかりで動作全体が進まない例

```
Fixpoint list_check (E1:Event)(LE:list Event):bool :=
 match LE with
 |nil ⇒ false
 |E2::TLE ⇒ Eeq_dec E1 E2 ||(list_check E1 TLE)
 end.
```

Eeq_dec とは bool 型の関数で，2 つの Event を比較し，一致していた場合は真を返すものである．“::” はリスト構成子を示す．A::B は A がリストの先頭要素，B が残りの要素のリストである．A::B::C は A::(B::C) の省略形である．match with とはパターンマッチングであり，これを使って list Event 型の LE において，動作の分岐を書くことができる．この分岐の優先度は，一番上が最も高く，下に向かうにつれ優先度が低くなるように設定される．また，非決定的な記述はできない．パターンマッチングを定義する際に，LE が取り得るすべての種類において分岐を記述しないとエラーが出る．

4.4 マッチング

以下のプログラムは，Interest パケットのマッチングに対する定義である．マッチングが成功するか否かで真偽値を返す関数 CSmat，PITmat，FIBmat で表現した．

```
Definition Interest_Match(N1 N2:Node)
            (LE:list Event):list Event:=
if CSmat(Entry N2 LE)
then (
    if list_check (Say N2 Data) LE &&
       list_check(AddStruct N2 (PIT N1)) LE
    then LE
    else Say N2 Data::
         AddStruct N2(PIT N1)::LE
     )
 else
 if PITmat (Entry N2 LE)
 then (
    if list_check (AddStruct N2 (PIT N1)) LE
    then LE
    else AddStruct N2 (PIT N1) :: LE
     )
 else
 if FIBmat (Entry N2 LE)
 then Say N2 Interest ::
        AddStruct N2(PIT N1)::LE
 else LE.
```

Interest_Match は，2 つの Node 型，1 つの Event 型リストから Event 型リストへの関数である．例えば，CSmat が false，PITmat が false，FIBmat が true であった場合，これは FIB でマッチングしたということに相当する．この場合，それまでのイベントの時系列リストの先頭に，Say イベントで Interest パケットの送信と，AddStruct イベントで PIT を加えるという 2 つのイベントを追加する．

Data パケットがマッチングするときは PIT でマッチングする場合以外はすべて処理が同じなので，同様にパターンマッチングを使わずに以下のように表現した．

```
Definition Data_Match (N : Node)
            (LE:list Event):list Event :=
if (PITmat (Entry N LE))
then(
    if list_check (Say N Data) LE &&
       list_check (AddStruct N CS) LE
    then LE
    else Say N Data :: AddStruct N CS :: LE
     )
else LE.
```

パケットや各構成要素は本来コンテンツ名を持つが，本研究ではコンテンツ名の定義を無視している．今回は CS が蓄えられるパケット数を有限としていないためコンテンツ名が複数あっても，異なるコンテンツ名における CCN の動作に互いに影響を及ぼさない．したがって，本研究ではコンテンツ名を 1 種類のみ扱うことにしている．

4.5 CCN のプロトコル

関数 Ccn は，CCN のプロトコルであり，リストの Event 型を取り，命題の型を返す．一行目の “→” はこれを意味している．

```
Inductive Ccn : list Event → Prop :=
|Cnil : Ccn (AddStruct 1 CS :: nil)
|R_ccn : forall(LE : list Event)(N : Node),
Ccn LE
→ 1 <N
→ In (AddStruct N FIB) LE
→ ~ In (Say N Interest) LE
→ ~ In (AddStruct N CS) LE
→ Ccn ((Say N Interest) :: AddStruct N (PIT N) :: LE)
|F_ccn : forall(LE : list Event)(N : Node),
 Ccn LE
→ In (Say N Interest) LE
→ Ccn ((Interest_Match N (parent N) LE))
|B_ccn : forall(LE : list Event)(N : Node),
Ccn LE
→ In (Say (parent N) (Data)) LE
→ In (AddStruct (parent N) (PIT N)) LE
→ Ccn ((Data_Match N LE)).
```

一行目以外の “→” は “ならば” を表す．“In” は要素がリストに含まれることを示す関数で，“In A B” は，リスト B は A を要素に含むことを表す．

イベントリスト LE に対して，“Ccn LE” は LE が CCN のプロトコルにしたがって動作したものであることを表す．すべてのイベントはどの時刻でも起こりうるものであり，イベントによってノードの構成要素が変化する．マッチングは構成要素に依存して結果が変わるため，イベントの起

こる順序によって各ノードのパケットに対する動作が変わる．したがって，イベントリストは一意に定まらない．

Ccn は初期状態 Cnil，ユーザからの通信の開始 R_ccn，Interest パケットの受信 F_ccn，Data パケットの受信 B_ccn という 4 つのコンストラクタをもつ．

このうちの 1 つ，F_ccn について説明する．イベントリスト LE はこのプロトコルにしたがって動作したものであり，ノード N が Interest パケットの送信をするというイベント (Say N Interest) がイベントリスト LE の要素に含まれているならば，ノード N と送信先のノード parent N 間でマッチングをする．マッチングは関数 Interest_Match を呼び出しておこなう．得られるイベントリスト (Interest_Match N (parent N) LE) は CCN のプロトコルにしたがって動作したものである．

5. Coq による検証

正当性は，様々な保証を定式化したものの複合体として示される．この保証の選び方には決まりが無く，検証する目的に応じて選択する必要がある．ここでは，与えられたシステムの動作の正当性を，システムの入力に対して正しい出力を生成することとする．本研究では，これを通信に関して当てはめて以下の 2 つを正当性とした．

(1) あるノードが要求を出せば，そのノードに対して正しい応答が返ってくるということ

(2) あるノードが応答を受け取るならば，そのノードは要求を出していたということ

(1) を Forward，(2) を Backward と呼ぶことにし，この 2 つに細かい必要条件を加味してそれぞれ以下のように記述した．これらを証明することで，完全二分木のネットワークトポロジであれば, ネットワークの規模に関係なく，通信が成功するといえる．

5.1 Forward

Forward の素直な記述は次のようになる．イベントリストが CCN のプロトコルにしたがって動作しており，「ユーザ N が，隣接する全ノードに Interest パケットを送信する (Say N Interest)」というイベントがリストの先頭にあり，N がサーバでない (1<N) ならば，いつか「N は Data パケットを受信する (Say N Data)」というイベントが起こる (図 11)．Coq のコードは以下のようになる．

```
Lemma Forward :
forall(LE : list Event)(N : Node),
Ccn((Say N Interest) :: LE)
→ 1 < N
→ exists LE1,
Ccn(((Say N Data):: LE1 ++ ((Say N Interest) :: LE))
```

“++”はリスト連接子を示し，“A++B” は，リスト A とリスト B を連結した 1 つのリストを表す．

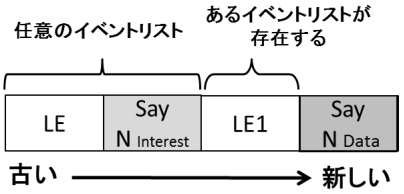

図 11 Forward

5.2 問題点と解決策

本モデルでは局所的に起こる複数のイベントを 1 本のイベントリストで管理しているため，異なる 2 つ以上のノード間で同時にイベントが起こることはなく，イベントリストに加えるイベントの順番は非決定的である．例えば，あるノード N1 からサーバへの送信イベントが起こった直後，N1 よりも下流のノード N2 から上流への送信イベントが起こることもあり，有限時間内で必ずしも CCN のプロトコルにしたがってパケットが上流に向かうとは限らない (図 12)．

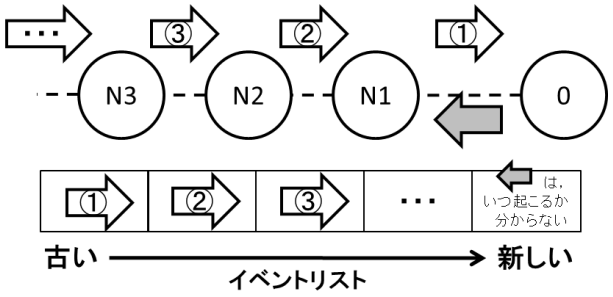

図 12 問題点

Forward を図 11 のように形式化すると，いつか「ユーザは Data パケットを受信する」というイベントが起これば良いので，別のイベントが割り込まない最短のイベントリストが起こったという 1 つの場合を証明すれば，証明は成功する．しかし，これではすべての場合を証明したことにはならない．この問題を解決するために，Forward の素直な記述の結論部を以下のように修正した．

- 有限長のイベントリストが先頭に追加される間に「ユーザ N が Data パケットを受信する (Say N Data)」イベントが起こるか，その後，別のイベントが割り込まない最短のイベントリストが起こった後に，「ユーザが Data パケットを受信する (Say N Data)」イベントが起こる (図 13)

対応する Coq のコードは以下のとおりである．

```
Lemma Forward2 :
forall (LE1 LE : list Event)(N : Node),
Ccn(LE1 ++ ((Say N Interest) :: LE))
→ 1 < N
→ In (Say N Data) LE1
∨ exists LE2, Ccn(((Say N Data):: LE2)
++ LE1 ++ ((Say N Interest) :: LE)).
```

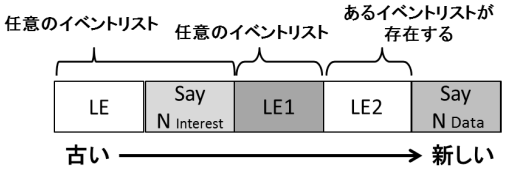

図 13　修正した Forward

5.3 Backward

Backward の記述は次のようになる．イベントリストが CCN のプロトコルにしたがって動作しており，ユーザ N がサーバでなく (1<N)，「N が Data パケットを受信する (Say N Data)」というイベントがリストの先頭にあるならば，以前に「ユーザが Interest パケットを送信する (Say N Interest)」というイベントが起こっていたはずである (図 14)．Coq のコードは以下のようになる．

```
Lemma Backward :
forall(LE1 : list Event)(N : Node),
Ccn ((Say N Data)::LE1)
 → 1 < N
 → In (Say N Interest) LE1.
```

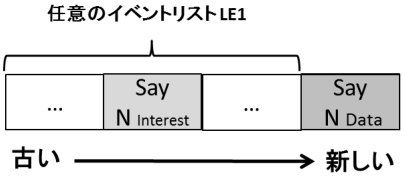

図 14　Backward

先に述べたように，イベントとはパケットの送受信 (Say) と，構成要素の追加 (AddStruct) のことである．マッチングが起こるとイベントが発生し，そのイベントが 1 本で管理されたリストの先頭に追加されるため，リストの先頭に近い方が新しく起こったイベントとなる．したがって，Backward の性質を証明するには，上記のような補題を証明すれば十分である．

5.4 証明

Forward の証明には，木の高さに関する帰納法を適用した．木の高さを Height という自然数として定義し，Node 型を持ち，Height 型を返す関数 my_height を定義した．補題には，木の高さを表す変数が含まれていないので，Height に関する帰納法を適用するために補題の仮定としてノード N に対してその高さを与えてくれるような式を追加する．補題を証明する際に，my_height の引数となるノード N と Forward の仮定に元から含まれる任意のノード N を関連付けて，my_height N を仮定に呼び出した．

Backward の証明にも，CCN のプロトコルに関する帰納法を適用した．関数 Ccn の 4 つのコンストラクタの各場合を証明することになる．4.5 節で示した Ccn の定義に基づき，イベントリスト LE を，C_nil の場合は AddStruct 1 CS :: nil に，R_ccn の場合は (Say N Interest) :: AddStruct N (PIT N) :: LE に，F_ccn の場合は (Interest_Match N (parent N) LE) に，B_ccn の場合は (Data_Match N LE) にそれぞれ展開して証明していく．

モデルと証明すべて合わせて，約 3500 行のプログラムとなり，約 80 個の補題で証明できた．

6. 議論

今回提示したモデルの有効範囲について考察する．

現在のモデルでは，各ノードが最初から常に FIB を持つとし，コンテンツ名は 1 種類しか考慮していない．FIB を持つノードと持たないノードを考える場合，Coq では，非決定的な関数定義は許されないため，現在のモデルを，通信が始まる前に FIB を配信するという過程を取り入れたモデルに変更する必要がある．このようにモデルを変更すると，コンテンツ名が複数存在する場合でも，そのコンテンツ名ごとに FIB の有無を表現できる．

また，コンテンツ名を複数個扱う場合，現在のモデルでは，すべてのイベントを 1 本のイベントリストで管理しているため，複数のコンテンツ名に対するイベントを同じリストで管理することになる．特定のノードに関係するイベントのみを抽出することに成功しているので，この 1 本のリストを，各コンテンツ名のイベントリストに分割するのは，可能であると推測する．

しかしながら，各コンテンツに分けて管理した複数のリストから，1 本のリストに統合しようとすると，そのリストが CCN のプロトコルを満たすとは限らない可能性がある．CCN のプロトコルを満たすということは，マッチングで追加されるイベントが正しい順序で並んでいるかということであり，各マッチングごとに追加するイベントは 1 つではなく，送受信イベント Say と構成要素を追加するイベント AddStruct の 2 つを同時に追加することもある．1 つのコンテンツ名に関して，あるイベントが 2 つ起こったときに，別のコンテンツ名に関するイベントが割り込む可

能性があり，それは CCN のプロトコルを満たすとは，現在のモデルではいえない．

あるノードの子が片方欠けているものや，両方とも欠けているような二分木の場合，欠けているノードを FIB を持たないノードだと表現すると，完全二分木のモデルがそのまま適用できる．また，同様の理由でこのモデルはライン型のトポロジにも適用できる．逆に，3 つ以上の子ノードを持つ木構造の場合は，このモデルはそのままでは適用できない．現在のモデルでは，親ノードを N/2，子ノードを 2∗N，2∗N+1 と考えることでノード間の接続関係を表現し，この関係を使って証明しているが，3 つ以上の子ノードを持つノードがあると，このように表現できないためである．同様の理由で，完全二分木のサーバの位置を変えると，子の数が 3 つになり，接続関係の規則が崩れてしまうため証明できるとはいえない．

7. 結論

本研究では，ネットワークアーキテクチャである CCN のプロトコルを，証明支援系 Coq を用いて帰納的にモデル化し，各ノードでおこなわれているマッチング処理を実装した．今回対象としたネットワークトポロジは完全二分木であり, ノード数を固定しないモデルである．また 1 対複数の送信を考慮し，Interest パケットの 3 種のマッチング，Data パケットの 3 種のマッチングが全て含まれている．そのモデル上で，動作の正当性の 1 つとして，ユーザがあるコンテンツを要求すれば，必ず正しいものを受信できることを検証した．

今回のモデルは, すべて合わせて約 3500 行のプログラムとなり，約 80 個の補題がある．これらの証明の戦略は主に帰納法と場合分けで構成される．マッチングの処理のところで，同じ場合分けが繰り返されることがあり，これを洗練すれば行数をもっと短くできると思われる．

今後は，FIB が必ずしもコンテンツ名を持たない場合や，コンテンツ名を複数個扱えるようにモデルを拡張する．

参考文献

[1] Bertot, Y. and Castŕan, P., “Interactive Theorem Proving and Program Development - Coq’Art: The Calculus of Inductive Constructions,” Springer Verlag, 1998.

[2] Bharadwaj, R., Felty, A., Stomp, F., “Formalizing Inductive Proofs of Network Algorithms,” Proceedings of the 1995 Asian Computing Science Conference, 1995, pp.335-349.

[3] Clarke, E. M., Grumberg, O. and Peled, D. A., “Model Checking,” MIT Press, 1999.

[4] Dai, H., Wang, Y., Fan, J., Liu, B., “Mitigate DDoS Attacks in NDN by Interest Traceback,” IEEE INFOCOM 2013 Workshop on Emerging Design Choices in Name-Oriented Networking, 2013, pp.381-386.

[5] Jacobson, V.，Smetters, D. K.，Thornton, J. D.，Plass, M. F.，Briggs, N. H. and Braynard, R. L.，“Networking Named Content ,” ACM CoNEXT, 2009, pp.1-12.

[6] Kaufmann, M., Monolios, P. and Moore, J. S., “Computer-Aided Reasoning an Approach,” Kluwer Academic Publishers, 2000.

[7] Kobayashi, N., “Model-Checking Higher-Order Functions,” 11th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming(PPDP09), 2009, pp.25-36.

[8] McMillan, K., “Symbolic Model Checking”, Springer Verlag, 1993.

[9] Nipkow, T., Paulson, L. C., and Wenzel, M., “Isabelle/HOL: A Proof Assistant for Higher-Order Logic,” Springer Verlag, 2002.

[10] Norell, U., “Dependently typed programming in Agda,” Advanced Functional Programming 2008, 2008, pp.230-266.

[11] Owre, S., Rushby, J. M. and Shankar, N., “PVS: A prototype verification system,” 11th International Conference on Automated Deduction, 1992, pp.748-752.

[12] Stewart, G., “Computational Verification of Network Programs in Coq,” Certified Programs and Proofs 2013, 2013, pp.33-49.

[13] 森嶋崇，後藤瑞貴，高橋和子，“証明支援系 Coq を使った CCN のモデル化と検証について，” 電子情報通信学会技術研究報告，Vol.114，No.21，pp.37-42.