Vol.2018-CLE-24 No.11
2018/3/21

Tremaを用いたSDN構築演習のための通信動作可視化システムの開発

浅野 晶文[1,a)]　立岩 佑一郎[1]　金 鎔煥[1]　片山 喜章[1]　新村 正明[2]

概要：本稿では，大学生向けに展開される Trema を用いた仮想ネットワーク上での SDN の構築演習を対象とする．この演習では，SDN での通信データの取り扱いの理解とコントローラをプログラミングする能力の習得を目的としている．演習問題では，達成条件の 1 つとして，指定された経路での ICMP エコーによる通信データの伝達が与えられる．出来上がったネットワークが達成条件を満たさないとき，学習者は当該通信データの伝達経路やコントローラの実行文，及びそれらの関係性を手がかりに誤りを絞り込む．この際，パケットキャプチャツールやステッパー等を適切に扱ってデータを集め，その結果を分析することで手がかりを得られる．しかし，それらを扱い手がかりを得ることは演習目的の範囲外であり，多くの学習者にとって実践が困難な作業である．また，人手での実践に時間がかかり，指導者による実践は学習者の待機時間が増大する．本稿では，誤りを絞り込む上記 3 つの手がかりを自動生成するシステムを提案する．そして，手がかりの素となるデータの自動収集法，そのデータから手がかりへの変換方法，誤りの絞り込みに効果的な手がかりの表現について述べる．

A development of communication activity visualizing system for SDN construction exercises using Trema

Asano Akifumi[1,a)]　Tateiwa Yuichiro[1]　Kim Yonghwan[1]　Katayama Yoshiaki[1]
Niimura Masaaki[2]

1. はじめに

ネットワークでの通信をソフトウェアによって動的に制御する Software-Define Network(以下，SDN) というネットワーク技術が普及してきている．それに伴って，大学等の教育機関では，SDN によるネットワーク構築演習 (以下，構築演習) が行われるようになってきている [1] [2].

構築演習では，SDN を実現する通信プロトコルの 1 つとして普及している OpenFlow[3] を採用している．OpenFlow は従来のネットワーク機器における通信データの転送機能とシステム制御機能を分離し，システム制御機能を OpenFlow コントローラ (以下，コントローラ) という 1 つの制御用プログラムで実現し，通信データの転送機能を OpenFlow スイッチというコントローラに対応するネットワークスイッチを複数用いて実現している．これらにより，ネットワークでの通信を 1 つのプログラムによって動的に変更することを可能にしている．構築演習では，ネットワーク構成とネットワークが達成すべき条件が与えられており，そのために利用するコマンドについても提示されている．達成すべき条件はホスト間での通信データの伝達である．構築演習はコントローラのプログラムを学習者にプログラミングさせる形式を取る．コントローラのプログラミングは以下の手順によって行われる．

手順 1　コントローラのプログラムの設計・コーディングを行う．

手順 2　コントローラをデバッグするためのテストを設計する．

手順 3　テストを実行する．

1　名古屋工業大学
Nagoya Institute of Technology
2　信州大学
Shinshu University
a)　asano@moss.elcom.nitech.ac.jp

手順 4　手順 3 の結果からコントローラの誤りを特定する．

手順 5　コントローラの誤りを修正し，手順 3 に戻る．

学習者は与えられたネットワーク構成において上記の手順 2，3 を行う．また，手順 2，3 のテストにおいては，提示されているコマンドを利用してホスト間での通信データの伝達を行ってその様子を確認する．ここで手順 4 の誤りの特定では，学習者は手順 3 の結果から，（1）当該通信データの伝達経路や（2）コントローラで実行された実行文，及び（3）それら 2 つの関係性を手がかりに誤りを絞り込んでいく．これら手がかりを得るには，手順 2，3 のテストの一部としてパケットキャプチャツールやステッパー等の利用手順を設定して実行しデータを集めておく必要がある．また，手順 4 で集めたデータを分析する必要もある．しかし，以下の問題点がある．

問題点 1　それらツールを扱ってその結果を分析し手がかりを得るには，それらツールの仕様を学習者が理解している必要がある．このことは構築演習の目的である SDN における通信データの取り扱いについての理解と，コントローラをプログラミングする能力の習得の範囲外であり，また，これらツールをテストに組み込むことはテストの設計・実行を複雑にし，多くの学習者にとって実践が困難な作業である．

問題点 2　人手での実践に時間がかかるため，指導者による実践では学習者の待機時間が増大する．

本稿では，上記の問題を解決するため，誤りを絞り込む上記 3 つの手がかりを自動生成するシステムを提案する．提案システムの特徴を以下に示す．

特徴 1　手順 3 を行う際に手がかりを得るための素となるデータを自動で収集し，その結果を分析したものを表示する．これにより，学習者が困難な作業を行わずとも手がかりを得られるようにする．また，学習者の待機時間も削減される．

特徴 2　手がかりを図で表現する．これにより，学習者が誤りの絞込みを効果的にできるようにする．

2. 関連研究

Trema における既存の実行ログの自動収集システムとしては Tremashark[4] がある．Tremashark は Wireshark のプラグインとして実装されているコントローラのデバッグの支援を目的としたシステムである．このシステムでは，ネットワークを構成するホスト・スイッチから出力された Syslog，ホスト・スイッチで伝達された通信データ，コントローラとスイッチ間で伝達された OpenFlow メッセージの送信と受信の情報，標準入力からのテキスト情報を収集し，それらの情報を 1 つに集約して時系列に並べたファイルを作成する．このファイルを Wireshark で読み込むと，収集された情報を一連の流れとして確認することができる．パケットとそのほかの情報を自動収集し，1 つにして時系列で整理する点，それを一連の流れとして確認できる点で本研究と同様であるが，コントローラの実行文の情報を収集しない点で本研究と異なり，第 1 章の手がかり（2）を得るには不十分であり，1 章で示した問題点 1 を解決できない．

3. SDN 構築演習

3.1 演習の概要

本稿で想定している SDN の構築演習は，大学の工学部情報系の演習授業であり，信州大学にて実践されている演習 [1] をモデルとする．学習者は数台のネットワーク機器によって構成されるネットワーク構成と達成条件を与えられており，そのネットワーク構成において，達成条件が満たせるようにコントローラをプログラミングする．与えられる達成条件は以下である．

条件 1　与えられているホスト全てで ICMP エコーによる通信データの伝達ができること．

条件 2　通信データが消失しないこと．

また，ネットワーク構成の情報として以下が与えられている．

(1) 各ホストのホスト名
(2) 各ホストの IP アドレス
(3) 各スイッチのスイッチ名
(4) 各ホストと各スイッチのポートとの配線接続関係

この演習は Linux の Network Namespace[5]，Open vSwitch[6]，OpenFlow のフレームワークである Trema[7] を利用して行う．学習者は Trema を利用するにあたり，コントローラのプログラムを Ruby[8] で記述する．この演習ではネットワークを構成するためのシェルスクリプトファイルが学習者に与えており，学習者はこのシェルスクリプトを実行することで仮想ネットワーク環境を用意する．学習者がこの演習において，テストを実行する場合には，以下の手順で行う．

手順 1　シェルスクリプトを実行して仮想ネットワーク環境を設定する．

手順 2　Trema の run コマンドを実行してコントローラを実行する．

手順 3　ICMP エコーを行うコマンド (ping) を任意の回数だけ実行する．

手順 4　Trema の killall コマンドを実行してコントローラを終了する．

上記手順で利用する各コマンドの使い方については指導者より提示されるとする．また，Open vSwitch に関する以下のコマンドについても指導者より提示される．

- **ovs-ofctl show スイッチ名** スイッチの状態の確認
- **ovs-ofctl dump-flows スイッチ名** フローテーブルの確認
- **ovs-ofctl help** スイッチに対するコマンドの一覧を表示

対象とする学習者は，TCP/IP や VLAN，ルーティング技術について学習済みであり，当該授業等において事前にネットワークの構築を学習するものとする．また，Ruby によるプログラミング経験や SDN によるネットワークの構築については経験がないものとする．演習は以下を目的とする．

目的 1　SDN における通信データの取り扱いについての理解

目的 2　コントローラをプログラミングする能力の習得

3.2　想定する誤りの特定手順

本稿で想定している誤りの特定手順では，本章 1 節で示したテスト手順を行い，その結果から誤りの特定を行う．本章 1 節のテスト手順を行った場合，まず，学習者は通信データが送信先に到達しないことから誤りの有無を認識する．ここで，学習者は誤りを特定するために 1 章で示した手がかり (1)，(2)，(3) を得ようとする．そのためには以下の手順を行うことが考えられる．

手順 1　本章 1 節のテスト手順 1 を行う．

手順 2　各ホスト・スイッチが持つネットワークインターフェースに対してパケットキャプチャを実行し，ホスト・スイッチ間を流れるパケットをキャプチャする．

手順 3　ローカルループバックアドレスでフィルターをかけてパケットキャプチャを実行し，コントローラとスイッチの間を流れる OpenFlow メッセージをキャプチャする．

手順 4　各文が実行されたことが分かるような標準出力を行うプログラムに書き換え，そのプログラムで本章 1 節のテスト手順 2〜4 を行う．

手順 5　手順 4 によって得られる標準出力とプログラムを見比べて手がかり (2) を得る．

手順 6　手順 2 のキャプチャ結果を確認し，同じパケットの取得地点とその取得時間を比較して伝達方向を分析する．また，手順 3 のキャプチャ結果を確認し，その OpenFlow メッセージの種類，取得時間を確認する．さらに，OpenFlow メッセージの取得時間順と手がかり (2) を比較して伝達方向を分析する．これらにより，手がかり (1) を得る．

手順 7　手がかり (1)，(2) を時間順で比較し，手がかり (3) を得る．

また，手がかり (1)，(2)，(3) から誤りを特定するには以下の手順を行うことが考えられる．

手順 1　手がかり (1) からどの地点での通信データの伝達が誤りにつながっているかを特定する．

手順 2　手順 1 の地点と手がかり (2)，(3) から，その地点に関わっているであろうプログラムの動作を特定し，その動作を実行した文を特定する．

手順 3　その文を確認し誤りを特定する．

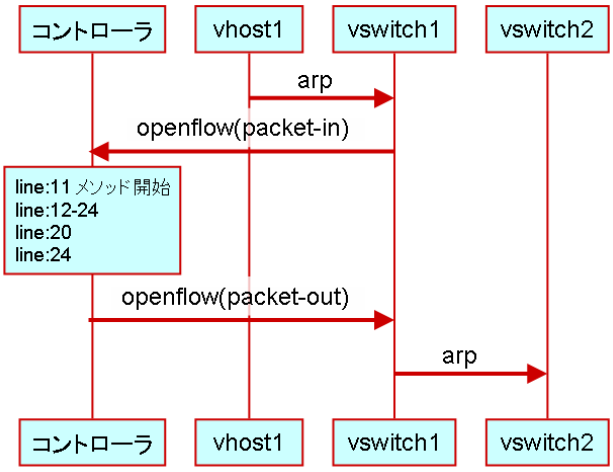

図 1　手がかりの表現の例

4.　提案する手がかり表現法

4.1　表現の構成

本稿で提案する手がかりの表現法では，以下の情報を図として表現する．

情報 1　ホスト，スイッチ，コントローラの間で送受信された通信データ

情報 2　コントローラで実行された文

情報 3　情報 1，2 の実行順序

図 1 の例では，ネットワークを構成するネットワーク機器である vhost1 という名前を持つ 1 台のホストと，vswitch1，vswitch2 という名前を持つ 2 台のスイッチにおける通信データの伝達の一部を表現している．図 1 の例では，通信データの伝達とコントローラでの文実行は以下である．

動作 1　vhost1 から vswitch1 に arp による通信データ (a) が伝達される．

動作 2　vswitch1 からコントローラに OpenFlow による通信データ (Packet-In メッセージ)(b) が伝達される．

動作 3　コントローラでプログラム中の 11 行目，12〜14 行目，20 行目，24 行目が順に実行される．

動作 4　コントローラから vswitch1 に OpenFlow による通信データ (Packet-Out メッセージ)(c) が伝達される．

動作 5　vswitch1 から vswitch2 に arp による通信データ (d) が伝達される．

4.2　表現の実現法

提案する表現法では，ノード，ベクトル，リストの 3 つのオブジェクトを用いて表現する．本章 1 節の情報 1，3 はノードとベクトルを用いることで表される．また，情報 2 はリストを用いることで表される．

ノードは演習問題で与えられているネットワークを構成するホスト，スイッチなどのネットワーク機器と，コントローラを表現するオブジェクトである．ノードは上下の四

角形とそれを結ぶ線で表現される．ノードは以下の要素を持つ．

- ノード名：コントローラ及びネットワーク機器の識別名を表す．
- ライン：時間の流れを表す．ラインに配置されるベクトル，リストはラインの上から下に向かって時間順になるように並ぶ．

ノードは以下の項目に従って，図に配置する．

(1) ノード名を四角形内部に示す．
(2) ラインを四角形を結ぶ線で示す．
(3) 図の上下に横並びで配置する．
(4) “コントローラ” をノード名とするノードを左端に置き，その他のノードは左から時系列昇順で配置する．

図 1 の例では，ノードとして表現される対象は，コントローラと vhost1，vswitch1，vswitch2 の 3 つのネットワーク機器である．3 つのネットワーク機器は本章 1 節の動作において，vhost1，vswitch1，vswitch2 の順に登場している．したがって，項目 (3) に従い，コントローラ，vhost1，vswitch1，vswitch2 の順に配置される．

ベクトルはコントローラ及びネットワーク機器間で伝達された通信データを表現するオブジェクトである．ベクトルは矢印で表現される．ベクトルは以下の要素を持つ．

- ベクトル名：通信データの arp，icmp などの分類を表す．
- 始点：通信データの送信元のコントローラ及びネットワーク機器を表す．
- 終点：通信データの送信先のコントローラ及びネットワーク機器を表す．
- 向き：通信データの伝達方向を表す．始点から終点に向けた方向である．

ベクトルは以下の項目に従って，図に配置する．

(1) ベクトル名を，矢印の上部に示す．
(2) 始点・終点をそれぞれ送信元・送信先と対応するノードのライン上とする．
(3) 始点から終点に向けて，ラインと垂直に配置する．

図 1 の例では，ベクトルとして表現される対象は，本章 1 節の動作 1，2，4，5 における通信データ (a)，(b)，(c)，(d) である．通信データ (a) は，vhost1 から vswitch1 に伝達している．したがって，ベクトルの始点と終点はそれぞれ，vhost1 と vswitch1 のそれぞれのノードのライン上になる．通信データ (b)，(c)，(d) についても同様に配置される．また，ベクトル名については，通信データ (a)，(d) は arp なので” arp”，通信データ (b)，(c) は OpenFlow の Packet-In，Packet-Out のメッセージなので，それぞれ” openflow(packet-in)”，” openflow(packet-out)” となる．

リストはコントローラで実行された文を先頭からの行数で表現するオブジェクトである．リストは線上の四角形で表現される．リストは以下の要素を持つ．

- 記述文：実行された文ごとの行数，またその文がクラス・メソッドの開始・終了定義かを実行された順で表す．1 つのリストにおける記述文の対象は，ある通信データの伝達から次の通信データの伝達までの間に実行された一連の文となる．

リストは以下の項目に従って，図に配置する．

(1) 記述文を四角形内部に示す．
(2) “コントローラ” をノード名とするノードのライン上に配置する．

図 1 の例では，リストとして表現される対象は動作 3 における一連の文実行である．

5. 提案システム

5.1 準備

本節では，本章で用いる共通の演算子，データ構造について述べる．

- 演算子 {}：引数を要素とする集合を返す．引数は可変長でカンマで区切る．
- 演算子＜＞：指定された順で引数を要素とする系列を返す．引数は可変長でカンマで区切る．例として＜ a，b，c ＞では，先頭が a，2 番目が b，3 番目が c の系列を返す．
- 演算子（）：引数を要素とする組を返す．
- 演算子 ．：組が持つ値へアクセスする．例として，組 A が要素 b を持つとき，b の値へのアクセスは “$A.b$” と記述される．
- 系列の要素 X_i：系列 X の i 番目の要素を X_i と表記する．
- 時刻：時間 h，分数 m，秒数 s，マイクロ秒数 ms による組（h，m，s，ms）で管理する．

5.2 システム構成

提案システムの構成を**図 2** に示す．提案システムは標準出力収集機能，キャプチャデータ収集機能，標準出力分析機能，キャプチャデータ分析機能，手がかり作成機能の 5 つからなり，またデータベースとしてログデータベースを用いる．

5.3 ログデータベース

ログデータベースでは，標準出力分析機能，キャプチャデータ分析機能から送られる実行文データ，パケットデータを格納する．実行文データ，パケットデータを格納するテーブルをそれぞれ実行文データテーブル（以下，LDT），パケットデータテーブル（以下，PDT）とする．この 2 つテーブルの定義をそれぞれ**表 1**，**表 2** に示す．これらの定義に従い，実行文データの組 $ldtd$ ＝（$time$，lst，led，$type$），パケットデータの組 $pdtd$ ＝（$time$，src，dst，$type$）で管理する．これらのテーブルのデータは，1 章で示した

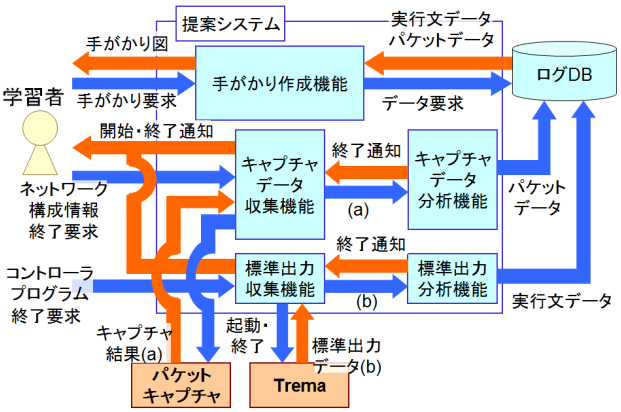

図 2　システム構成図

表 1　実行文データテーブルの定義

カラム名	データ型	説明
time	時刻	実行文の実行時刻
lst	整数型	実行文の開始行番号
led	整数型	実行文の終了行番号
data	文字列型	上記以外の備考情報

表 2　パケットデータテーブルの定義

カラム名	データ型	説明
time	時刻	パケットの到達時刻
src	文字列型	送信元のネットワーク機器名
dst	文字列型	送信先のネットワーク機器名
type	文字列型	パケットの分類

プログラミングの手順 3 を行う前にリセットされる．

5.4　標準出力収集機能

標準出力収集機能では，学習者からの Trema 用のコントローラの Ruby プログラムまたは終了要求を入力とする．

Ruby プログラムを入力とした場合，まず，ログデータベースの LDT のレコードを消去する．次に，Ruby プログラムの文を要素とし，それらをプログラムでの配置順で格納した系列 R から実行履歴収集用プログラム $R' = Deb(R)$ を作成する．関数 $Deb(R)$ の定義を**図 3** に示す．また，作成されるプログラムの例を**図 4** に示す．図 4 の例では，メソッドの開始・終了を含む 6〜11 行目の 6 行に対して標準出力の記述の追加を行っている．図 4 中の “（時間）” は “time.hour.to_s+“:”+time.min.to_s+“:”+time.sec.to_s+“.”+time.usec.to_s” である．この時，8〜9 行目は 1 つの文なので，その文の終了行の次の行に記述を追加する．次に，プログラム R' を用いて Trema コマンドを実行してコントローラを実行する．この際，Trema が出力する標準出力を監視して取得する．最後にコントローラ実行開始の通知を学習者へ出力する．

終了要求を入力とした場合，コントローラの実行を終了させる．その後，取得した標準出力を標準出力分析機能に入力として渡す．標準出力分析機能から終了通知が来た場合，終了通知を学習者へ出力する．

function Deb(系列 R)
　R'=<>
　for $i = 1$ to $|R|$ **do**
　　s'=“time=Time.new;puts(“［@dbg］line：+R_i の開始行+“-”+R_i の終了行+“,time：”+time.hour.to_s+“：”+time.min.to_s+“：”+time.sec.to_s+“.”+time.usec.to_s”
　　if R_i がクラス定義のヘッダである **then**
　　$s' = s'$+“+“,[class define]”)”
　　R' の末尾に R_i を追加
　　R' の末尾に s' を追加
　　else if R_i がメソッド定義のヘッダである **then**
　　$s' = s'$+“+“,[method define]”)”
　　R' の末尾に R_i を追加
　　R' の末尾に s' を追加
　　else if R_i がクラス・メソッドの終端である **then**
　　$s' = s'$+“+“,[end]”)”
　　R' の末尾に s' を追加
　　R' の末尾に R_i を追加
　　else
　　$s' = s'$+“)”
　　R' の末尾に R_i を追加
　　R' の末尾に s' を追加
　　end if
　end for
　return R'
end function

図 3　関数 Deb の定義

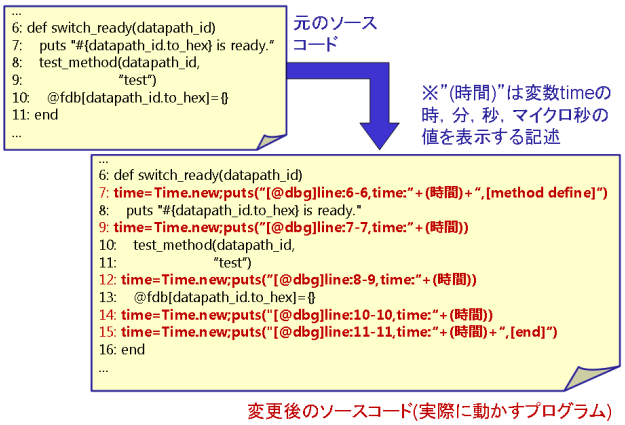

図 4　実行履歴収集用プログラム作成例

5.5　標準出力分析機能

標準出力分析機能では，プログラム R'_i による標準出力データ SW を入力とする．SW は，図 3 の s' により出力された文字列 str と，$stat$ の実行や Trema により出力される文字列を要素とし，それらが出力順に先頭から並んだ系列である．

まず，SW_i が先頭に “[@dbg]” を含む場合，以下を実行する．

(1) SW_i から $sline$=R_i の開始行，$tline$=R_i の終了行，tl=(time.hour.to_s，time.min.to_s，time.sec.to_s，time.usec.to_s) を抽出する．

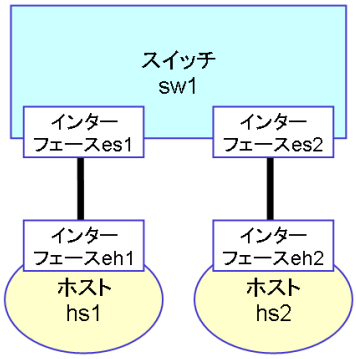

図 5 ネットワーク構成の例

(2) SW_i に “[class define]”，“[method define]”，“[end]” が含まれる場合，その文字列を d として抽出する．

(3) 表 1 の time = tl，lst = $sline$，led = $tline$，data = d として LDT にデータを追加する．

最後に終了通知を標準出力収集機能に出力する．

5.6 キャプチャデータ収集機能

学習者からのネットワーク構成情報 NWT または終了要求を入力とする．ネットワーク構成情報 NWT は，ネットワークインターフェース名 ni，ni のネットワークインターフェースを持つホストまたはスイッチの名前 na，na とリンクされているホストまたはスイッチの名前 nb による組（ni，na，nb）を要素とする集合で管理される．例として**図 5** に示すネットワーク構成の場合，NWT = {(es1, sw1, hs1), (es2, sw1, hs2), (eh1, hs1, sw1), (eh2, hs2, sw1)} となる．図 5 では，インターフェース名 eh1，eh2 のインターフェースをそれぞれ持つホストのホスト名がそれぞれ hs1，hs2，インターフェース名 es1，es2 のインターフェースを持つスイッチのスイッチ名が sw1 となっているネットワークが構成されている．

ネットワーク構成情報 NWT を入力とした場合，まずログデータベースの PDT のレコードを消去する．次に $nwt \in NWT$ において，$nwt.ni$ に対してパケットキャプチャを実行する．この際，対象のインターフェースが受信したパケットのみをキャプチャするように設定する．$nwt.ni$ へのキャプチャにおける結果は，時刻 tp にキャプチャしたパケットの分類情報 $ptype$ による組（nwt，$ptype$，tp）を要素とする集合 CA で管理される．次に OpenFlow メッセージをキャプチャするため，ローカルループバックのインタフェースに対してパケットキャプチャを実行する．このキャプチャにおける結果は，時刻 tof にキャプチャしたパケットの OpenFlow メッセージの種類情報 ofm，通信相手のスイッチ名 $ofsw$，スイッチが送信元なら 0，送信先なら 1 である情報 ofd による組（nwt，ofm，$ofsw$，ofd，tof）を要素とする集合 CB で管理される．

終了要求を入力とした場合，全てのパケットキャプチャを終了させ，CA，CB を得る．CA，CB をキャプチャデータ分析機能に入力として渡す．キャプチャデータ分析機能から終了通知が来た場合，終了通知を学習者へ出力する．

5.7 キャプチャデータ分析機能

キャプチャデータ分析機能では，本章 6 節の CA，CB を入力とする．

まず，$ca \in CA$ に対し，表 2 の time = $ca.tp$，src = $ca.nwt.nb$，dst = $ca.nwt.na$，type = $ca.ptype$ として PDT にパケットデータを追加する．次に $cb \in CB$ に対して以下を行う．

- $cb.ofd = 0$ の場合は，表 2 の time = $cb.tof$，src = $cb.ofsw$，dst = “コントローラ”，type = “openflow($cb.ofm$)” として PDT にパケットデータを追加する．$cb.ofd = 1$ の場合は，表 2 の time = $cb.tof$，src = “コントローラ”，dst = $cb.ofsw$，type = “openflow(”+$cb.ofm$+“)” として PDT にパケットデータを追加する．

最後に終了通知をキャプチャデータ収集機能に出力する．

5.8 手がかり作成機能

手がかり作成機能では，学習者からの手がかり要求を入力とする．

まずテーブル種別 $table$，開始レコード番号 i，終了レコード番号 j，time の値 $time$ による組（$table$，i，j，$time$）を要素とする系列 U を考える．系列 U は PDT の i 番目のレコードにより U_k =（“pdtd”，i，i，レコードの time の値）となる．また，LDT の i 番目のレコードにより U_l =（“ldtd”，i，i，レコードの time の値）となる．次に U を $time$ で昇順にソートした系列 U' を作成し，U' に対して以下を行う．

(1) $(U'_{i-1}.table \neq \text{“ldtd”}) \wedge (U'_i.table = \text{“ldtd”}) \wedge (U'_j.table = \text{“ldtd”}) \wedge (U'_k.table = \text{“ldtd”}) \wedge (U'_{k+1}.table \neq \text{“ldtd”})(i < j < k)$ となる i と k を見つける．すなわち，U' において実行文の要素の塊である．

(2) U'_i = (“union”，i，k，$U'_i.time$) とし，U' の $i+1$ から k までの要素を取り除く．

次に，U'_i を引数とする関数 $Drawf(u)$ を利用し 4 章 2 節で定義した図を作成する．手順を以下に示す．

(1) 図に “コントローラ” をノード名とするノードを配置し，そのノードにラインを配置する．

(2) U' の全ての要素 u に対して，先頭から順に $Drawf(u)$ を実行する．

関数 $Drawf(u)$ を**図 6** に示す．図 6 に関連した関数は以下である．

- $Record(table, i, col)$: $table$ はテーブル種別，i はレコード番号，col はカラム名．テーブル種別 $table$ の i 番目

```
function Drawf(U′ の要素 u)
  ND= {}
  if u.table=“pdtd” then
    if Record(u.table, u.i,“src”) ∉ ND である then
      Record(u.table, u.i,“src”) をノード名とするノードとそのラインを図に配置する．
      Record(u.table, u.i,“src”) を ND に追加する．
    end if
    if Record(u.table, u.i,“dst”∉ ND である then
      Record(u.table, u.i,“dst”) をノード名とするノードとそのラインを図に配置する．
      Record(u.table, u.i,“dst”) を ND に追加する．
    end if
    Record(u.table, u.i,“type”) をベクトル名，Record(u.table, u.i,“src”) を始点のノード，Record(u.table, u.i,“dst”) を終点のノードとするベクトルを図に配置する．
  else if u.table=“ldtd” then
    string=“line : ”
    if Record(u.table, u.i,“lst”) = Record(u.table, u.i,“led”) then
      string = string + Record(u.table, u.i,“lst”)
    else
      string = string + Record(u.table, u.i,“lst”)+“-”+Record(u.table, u.i,“led”)
    end if
    string = string + Record(u.table, u.i,“data”)
    string を記述文としたリストを図に配置する．
  else if u.table=“union” then
    list=<>
    for x = u.i to u.j do
      string=“line : ”
      if Record(u.table, u.x,“lst”) = Record(u.table, u.x,“led”) then
        string = string + Record(u.table, u.x,“lst”)
      else
        string = string + Record(u.table, u.x,“lst”)+“-”+Record(u.table, u.x,“led”)
      end if
      string = string + Record(u.table, u.x,“data”)
      list の末尾に string を追加する．
    end for
    list の要素を先頭から順に記述文としたリストを図に配置する．
  end if
end function
```

図 6 関数 $Drawf$ の定義

のレコードのうち，カラム名 col の値を返す．

最後に作成した図を手がかり図として，学習者に出力を返す．

5.9 システムの利用法

提案システムを用いた演習では，学習者は 3 章で示したものに加えてネットワーク構成情報が与えられているとする．3 章に示したテストの実行の手順のうち，手順 2，4 で学習者は Trema の代わりに提案システムを利用する．手順 2 では，学習者は Ruby で記述されたコントローラのプログラムソースコードとネットワーク構成情報を入力として提案システムを実行する．手順 4 では，学習者は入力として終了要求を提案システムに与えて提案システムを終了させる．誤りの特定を行う際に，学習者は任意で，手がかり要求を提案システムに与えることができる．その場合，学習者は手がかり図を得られる．

6. プロトタイプシステム

提案システムの標準出力収集機能，キャプチャデータ収集機能について，プロトタイプシステムを開発した．プロトタイプシステムの実装には Ruby を使用し，CUI として実装した．また，キャプチャデータ収集機能におけるパケットキャプチャでは tcpdump[9] を利用した．

プロトタイプシステムでは，Ruby で記述されたコントローラのプログラムソースコードとネットワーク構成情報を入力としている．また，Trema の killall コマンドを実行してコントローラを終了させることで，プロトタイプシステムが終了するようになっている．出力はコントローラからの標準出力を記録したテキストデータと，tcpdump による pcap 形式のキャプチャデータである．プロトタイプを利用した場合，3 章 1 節に示したテストの実行手順は以下のようになる．

手順 1 シェルスクリプトを実行して仮想ネットワーク環境を設定する．

手順 2 Ruby コマンドでプロトタイプシステムを実行する．プロトタイプシステムにより，Trema のコントローラと tcpdump が実行される．

手順 3 ICMP エコーを行うコマンド (ping) を任意の回数だけ実行する．

手順 4 Trema の killall コマンドを実行してコントローラを終了する．コントローラの終了を受けてプロトタイプシステムが tcpdump を終了させ，キャプチャ結果と標準出力のデータを出力する．

プロトタイプシステムを実行した実行画面を**図 7** に示す．図 7 のうち，上の端末ではプロトタイプシステムを実行し，下の端末では ping コマンドでホスト間通信を行った．プロトタイプシステムでは，ping コマンドが実行された際に，コントローラが出力した標準出力を表示させている．

また，5 章 8 節における図の作成では PlantUML[10] を用いる方法を採用する予定である．この方法では PlantUML への入力とするテキストファイルを作成し，PlantUML に渡し，出力として得られる図を手がかり図とする．この方法を採用する場合，5 章 8 節の図の作成手順は以下のようになる．

(1) テキストファイルの末尾に “@startuml”，“participant コントローラ as controller” を改行で区切って追加する．

(2) U' の全ての要素に対して，先頭から順に $Drawf(u)$

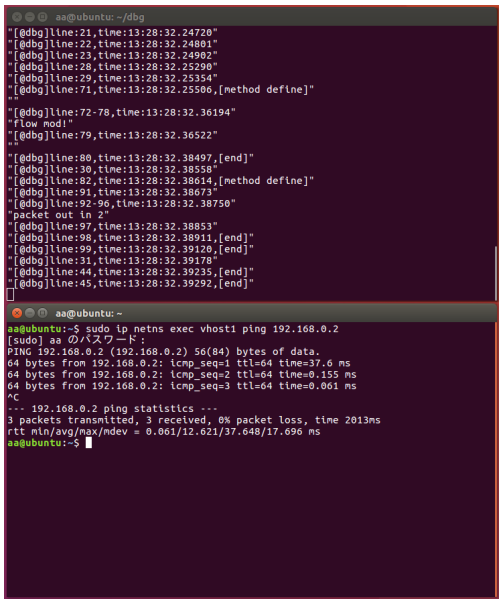

図 7　プロトタイプシステムの実行画面

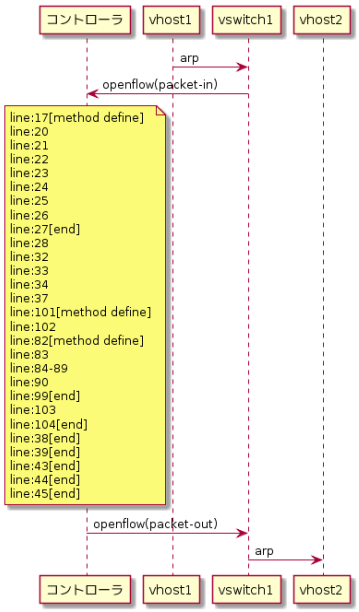

図 8　UML 図の例

を実行する．

(3) テキストファイルの末尾に “@enduml” を追加する．

また，図 6 に示した関数 $Drawf(u)$ の 4〜12 行目，21 行目，34 行目において，それぞれ以下を行う．

4〜12 行目　文字列 $sv = Record(u.table, u.i, \text{“src”}) + \text{“->”} + Record(u.table, u.i, \text{“dst”}) + \text{“:”} + Record(u.table, u.i, \text{“type”})$ を，テキストファイルの末尾に追加する．

21 行目　テキストファイルの末尾に “note over controller”，*string*，“endnote” を順に改行で区切って追加する．

34 行目　テキストファイルの末尾に “note over controller”，*list* の全要素，“endnote” を順に改行で区切って追加する．

プロトタイプの実行で得られた結果の一部から，未実装の機能の部分を人手で行いテキストファイルを作成し，PlantUML へ与えた場合の出力を**図 8** に示す．図 8 は vhost1，vhost2，vswitch1 からなる仮想ネットワークで vhost1 から vhost2 に ping コマンドで通信を行った場合の結果の一部である．

7. おわりに

本稿では，SDN の構築演習において，学習者が誤り特定のための手がかりとする，通信データの伝達経路やコントローラで実行された実行文，及びそれら 2 つの関係性を 1 つの図として表現する方法を提案した．また，この図を自動生成するため，それらの素となるデータを自動で収集し，そのデータを変換して図を自動生成するシステムを提案した．今後の課題として，標準出力分析機能，キャプチャデータ分析機能，手がかり機能の開発，表現法の効果性に関する評価実験などが挙げられる．

参考文献

[1] 新村 正明： 情報基礎特論 II，信州大学 (2017 年度).
[2] 長谷川 剛，ほか： 情報ネットワーク学演習 II，大阪大学 (2016 年度).
[3] ：Open Datapath Standardized Switch Protocol in Software Defined Network (SDN)(online)，入手先 ⟨https://www.opennetworking.org/projects/open-datapath/⟩ (2018.02.19).
[4] ：Tremashark(online)，入手先 ⟨https://www.slideshare.net/chibayasunobu/tremashark⟩ (2018.02.19).
[5] ：network_namespaces(7) - Linux manual page (online)，入手先 ⟨http://man7.org/linux/man-pages/man7/network_namespaces.7.html⟩ (2018.02.19).
[6] ：Open vSwitch(online)，入手先 ⟨http://www.openvswitch.org/⟩ (2018.02.19).
[7] ：Trema Full-Stack OpenFlow Framework in Ruby and C(online)，入手先 ⟨http://trema.github.io/trema/⟩ (2018.02.19).
[8] ：オブジェクト指向スクリプト言語 Ruby(online)，入手先 ⟨https://www.ruby-lang.org/ja/⟩ (2018.02.19).
[9] ：Manpages of TCPDUMP(online)，入手先 ⟨https://www.tcpdump.org/tcpdump_man.html⟩ (2018.02.19).
[10] ：シンプルなテキストファイルで UML が書けるオープンソースのツール，入手先 ⟨http://plantuml.com/⟩ (2018.02.19).