L-003

データベース定義型 SDN における Ryu コントローラを使用したネットワーク制御機能の実装
Network Management by using Ryu Controller in Database-oriented SDN Architecture

岩本 廉磨[1)]　佐藤 寧洋[1)]
Renma Iwamoto　Yasuhiro Sato

1　はじめに

既存のネットワーク制御では，各機器によって設定画面やコマンドが異なるため，管理者にとって設定変更による手間やコストが大きく，設定ミスによるネットワーク全体への影響も懸念される [1]．そのような背景から，ソフトウェアでプログラマブルにネットワーク制御を実現する Softwere-Defind Network（SDN）が提案されている．SDN によるネットワークでは，パケット転送に特化したスイッチと制御機能を担うコントローラで構成されており，物理的・論理的に分離されている．ネットワーク全体の状況を確認しながら，コントローラが複数のスイッチを一元的に制御することで，従来よりも柔軟な制御が実現している．しかし，原理的にスイッチがコントローラへ定期的な問い合わせをして負荷が集中するため，複数のコントローラによる分散制御が検討されているが，コントローラ間での情報同期や異種コントローラ間での制御が課題である [2]．我々の研究グループでは，ネットワーク全体の制御情報や接続状況などをすべてデータベース上で管理するデータベース定義型 SDN を提案している [3]．データベース定義型 SDN では，ネットワーク管理で必要な全ての情報をデータベースに集約することで，コントローラ間の情報同期の解決を目指している．抽象的な情報をデータベースに保持することで，実装の異なるコントローラ間の制御情報を一元化し，異種コントローラの混在を可能としている．様々なコントローラに対応するためには，抽象的なネットワーク情報から具体的な設定情報に変換する必要があるが，データベースに格納すべき情報やネットワーク制御情報の生成，実際の制御方法は検討されていない．

本稿では，SDN を実現するための技術の一つである OpenFlow プロトコルを対象とし，代表的な OpenFlow コントローラである Ryu のネットワーク設定方法を調査し，Ryu REST API の構成法について述べる．物理ネットワークの結線状態やスイッチの状態，管理者からの設定内容を格納するデータベースを設計・実装し，データベース言語によって取得・格納をする．そのデータベース言語をもとに，ネットワーク設定を HTTP 経由で送信ができる REST API を生成できるプログラムを実装する．動作確認のために実験ネットワーク上において，設定に基づいた通信ができるかを確認する．

2　データベース定義型 SDN

本節では，データベース型 SDN について概説する．

2.1　概要

図 1 にデータベース型 SDN のシステム概念図を示す．ネットワーク情報を格納するためのデータベースを中心として，経路や割当リソースなどの計算を担当するエンジン，コントローラに適合するための制御情報に変換するためのドライバで構成されている．最新の設定情報をデータベース内で管理することで，各コントローラ間のネットワーク情報の同期が取れるようになり，整合性を容易に保つことが期待できる．

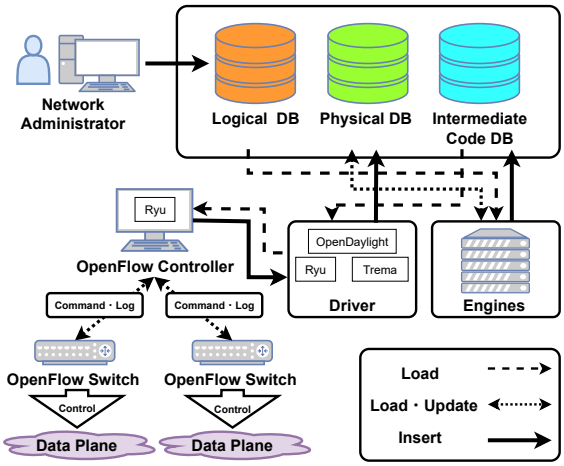

図 1: データベース定義型 SDN の概念図

2.2　データベース

`Logical DB` には，ネットワーク管理者やユーザが指定したネットワーク設定を格納する．`Physical DB` には，スイッチやパソコンなどのノードやリンクの接続状況，ネットワークの統計情報など物理的な接続情報を格納する．`Intermediate_Code DB` には，様々なコントローラに対応するために，特定の様式を持たない抽象的な設定情報を格納している．

2.3　エンジン

エンジンは，管理者などによって指定された抽象的なネットワーク設定情報を `Logical DB` から読み出し，`Physical DB` に格納されている物理ネットワークの接続情報をもとに，実際に制御するための経路計算やリソース計算・割当などを生成する．その後，様々なコントローラにも対応できるように中間的（抽象的）な制御情報を `Intermediate_Code DB` に格納する．

2.4　ドライバ

ドライバは，`Intermediate_Code DB` に格納されている情報をもとに，使用しているコントローラが認識できるコマンドや設定ファイルを生成する．一般的な動作例として，ネットワーク管理者が二台の端末を接続する設定を想定すると，`Intermediate_Code DB` からデータを取得して，二台の端末が接続されているスイッチを制御しているコントローラが認識できるデータに変換して，スイッチに送信する動作を担当する．

3　Ryu

Ryu は SDN 機能を提供するフレームワークで，OpenFlow や Netconf など様々なプロトコルに対応して

1）　大阪電気通信大学大学院工学研究科
Graduate School of Egineering, Osaka Electro-Communication University

表 1: 使用機材一覧

機材名	ベンダ	CPU	RAM	OS
PowerEdge R340	DELL	Xeon E-2288G	64GB	ESXi 7
N40	MINIS FORUM	Celeron N4020	4GB	Ubuntu 20.04
自作マシン	—	Core i5 12400	32GB	Ubuntu 22.04

おり，柔軟なネットワーク制御を容易にする API が提供されている [4]．REST API に対応したコントローラアプリケーションも提供されており，他のシステムと連携しやすくなっているのが特徴である．

コード 1 に Ryu で提供されている REST API の例を示す．この REST API では，Datapath ID（dpid）が 1 のスイッチに対して，該当する MAC アドレス（eth_dst）とその動作や出力ポート（actions）を指定している．データ形式は JSON であり，POST メソッドを使ってコントローラに対してデータを送信する．

コード 1: Ryu REST API による設定例

```
curl -X POST -d '{"dpid": 1,"priority": 10,"match
    ":{"eth_dst":"dc:a6:32:93:1f:4b"},"actions
    ":[{"type":"OUTPUT","port":5}]}'http
    ://192.168.45.2:8080/stats/flowentry/add
```

4 実験環境

本節では，本稿で使用した実験機材と実験ネットワークについて概説する．

4.1 実験機材

表 1 に，本実験で使用した端末の一覧を示す．DELL PowerEdge R340 には，VMware ESXi 7.0 をインストールし，データベースサーバの sql とエンジンサーバの Engines を仮想マシンとして構成した．sql と Engines には，Ubuntu server 22.04 をインストールしており，sql で使用しているデータベースソフトウェアは MySQL 8.0.32-0ubuntu0.20.04.2 とし，Engines で動作させたエンジンプログラムは Python で記述した．N40 の OS は Ubuntu Server 20.04 をインストールし，一台は OpenFlow コントローラとして Ryu をインストールした．もう一台は動作検証用の端末として使用した．自作マシンも実験ネットワークの動作確認用として使用する端末であり，Ubuntu 22.04 Desktop をインストールしている．OpenFlow スイッチには，Allied Telesis 社の CentreCOM AT-X230-10GT を使用し，動作させる OpenFlow バージョンは 1.3 とした．

4.2 実験ネットワーク

図 2 に，実験ネットワークの概要図を示す．コントロールプレーンのネットワークアドレスを 192.168.45.0/24 とし，Ryu コントローラ（of）やデータベースサーバ（sql），エンジン（Engines），管理者用端末などのネットワーク制御に使用する端末を接続した．データプレーンのネットワークアドレスを 192.168.10.0/24 と 192.168.20.0/24 とし，利用ユーザを想定した Host1 と Host2 を接続した．

5 実装

本節では，データベース定義型 SDN におけるデータベースと Engine プログラムの実装について概説する．

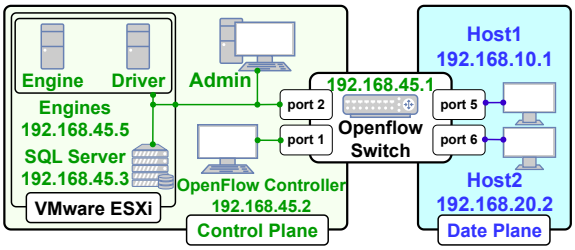

図 2: 実験ネットワーク図

5.1 概要

物理ネットワークの情報は管理者などが初期情報として静的に設定することを想定しているが，パケットの転送方法や処理などは動的に更新することになる．そのため，各スイッチから情報を収集したり，通信フローの状況を適宜確認しながら，必要に応じて設定を変更，すなわちデータベース内の情報を更新していく必要がある．本研究では，スイッチから情報を収集したり，経路情報を計算したりする処理とコントローラの実装に依存しない設定情報の生成をエンジンが担当する．エンジンが生成した抽象的な設定情報を使用しているコントローラに適した設定情報に変換するのがドライバである．

本稿では，データベース内の構成と管理者によるネットワーク設定，設定に基づいた転送ルールの作成，Ryu コントローラへの設定の反映の処理について実装する．本来であれば上記の通り，Ryu コントローラへの設定情報の変換および適用はドライバが担当する処理であるが，テンプレート化に向けた基礎検討として，すべてエンジンプログラムで処理を実施することとする．

5.2 データベースの構築

データベースは VMware 上にある仮想マシン sql に，Physical DB, Logical DB, Intermediate_Code DB の三つを作成した．各データベースには，それぞれ情報を格納するためのテーブルを定義作成する必要があるが，今回は主に物理ネットワークの接続状態やスイッチの情報を管理することが主となるため，Physical DB に必要な三つのテーブルを作成する．Interface_Traffic テーブルには端末の物理的な情報や MAC アドレス，出力ポート番号を格納している．Node テーブルには各端末のホスト名や端末が接続されているスイッチの Datapath ID を格納している．dpid には，ネットワーク内にあるスイッチに関する情報を管理しており，スイッチを特定するための Datapath ID などを格納している．

5.3 Engine プログラム

本節では，エンジンプログラムの実装について概説し，各データベースとの連携やコントローラへの設定方法などについて述べる．

5.3.1 概要

Engine プログラムは，各データベース（主に Logical DB や Physical DB）の情報をもとにネットワークの制御情報を生成し，Intermediate_Code DB に格納する．5.1 節で述べたとおり，本稿では Ryu コントローラへの設定法および REST API の構成法について検討するため，本来はドライバの処理であるコントローラへの設定処理についても Engine プログラム内で実行することとする．Ryu の REST API による設定で

は，HTTP の POST メソッドを使用することから，該当するコントローラへの URL と設定内容となる JSON 形式の設定データである．本稿では，主にこの URL の作成法と JSON 形式のデータの構成法について検討する．

5.3.2 Engine とデータベースの接続

Engine とデータベースの接続には，mysql-connector-python [5] を使用する．具体的にはコード 2 に示す Python コードによって実装した．データベースサーバの IP アドレスやデータベース名，パスワード，文字コードを設定し，SQL クエリを実行するための cursor オブジェクトを取得する．

コード 2: Engine とデータベースの接続

```
connection = mysql.connector.connect(
  host='192.168.45.3',
  user='py',
  passwd='**********',
  db='Physical',
  charset='utf8'
)
if connection.is_connected:
  cursor = connection.cursor()
```

5.3.3 REST API へのアクセス方法

REST API は URL 形式で，HTTP 通信を行う必要があるため，requests モジュール [6] を使用した．REST API によるアクセスでは，コントローラに対して設定を行う場合は，POST メソッド，取得する場合は GET メソッド，削除する場合は DELETE メソッドを利用する．

5.3.4 REST API の URL 生成

REST API の URL を生成する際に必要な情報をデータベースから取得して，その情報をもとに URL を構成するための処理を，コード 3 に示す．今回の実装において，設定対象のスイッチは管理者がホスト名で指定することとしているため，input 関数で取得している．データベースからは Node_ID，Node_name，Datapath_ID，Port，IP_ADDRESS が返ってくるので，それらの情報に基づいて URL を構成している．

コード 3: URL 生成処理

```
name = input()
sql = "select Node.Node_ID, Node_name, dpid,
     REST_Port, Interface_Traffic.
    Source_IP_Address from Node inner join
    dpid on Node.Node_ID = dpid.Node_ID
    inner join Interface_Traffic on dpid.
    Node_ID = Interface_Traffic.Interface_ID
     where Node.Node_name = %s;"
param = (name,)
cursor.execute(sql,param)
info = cursor.fetchone()
Node_ID = info[0]
Node_name = info[1]
Datapath_ID = info[2]
Port = info[3]
IP_Address = info[4]
url = 'http://' + IP_Address + ':' + Port +
    '/stats/flow/' + Datapath_ID
```

5.3.5 JSON 形式データの生成処理

コード 4 に，スイッチの設定に必要な情報をデータベースから取得して，その情報をもとに JSON 形式のデータを作成する処理を示す．input 関数で制御対象となる端末のホスト名を入力している．ホスト名をキーとしてデータベース内を検索し，設定に必要な MAC アドレスや出力ポート番号，接続されているスイッチを取得している．ここでは，指定した端末の MAC アドレスと出力ポート，そのときの動作（actions）を指定するフローエントリを作成している．

コード 4: JSON 形式データ生成処理

```
import json
name = input()
sql = 'select Node.Node_ID, Datapath_ID,
    Node.Node_name, Priority,
    Source_MAC_Address,
    Destination_MAC_Address, Source_Port
    from Node inner join Interface_Traffic
    on Node.Node_ID = Interface_ID where
    Node.Node_name = %s;'
param = (name,)
cursor.execute(sql,param)
info = cursor.fetchone()
flow_entry = dict()
match = dict()
action = dict()
actions = []
Node_ID = info[0]
Datapath_ID = info[1]
Node_name = info[2]
Priority = info[3]
Source_MAC_Address = info[4]
Source_Port = info[6]

match['eth_dst'] = Source_MAC_Address
action['type'] = 'OUTPUT'
action['port'] = Source_Port
actions.append(action)
flow_entry['dpid'] = Datapath_ID
flow_entry['priority'] = Priority
flow_entry['match'] = match
flow_entry['actions'] = actions
conf = json.dumps(flow_entry)
```

5.3.6 ブロードキャストパケットの処理

コード 5 に，ブロードキャスト宛のパケットを処理するための設定方法を示す．宛先 MAC アドレスが不明な場合や，ARP リクエストなどはブロードキャスト宛のパケットとして送信されるので，ブロードキャストパケットを処理するためのフローエントリを作成している．受信したフレームの宛先 MAC アドレスがブロードキャスト（ff:ff:ff:ff:ff:ff）であれば，全ポートに出力（フラッディング）するという設定を行うためのデータである．5.3.4 で示した URL の引数データとして，以下のコードで生成された JSON 形式のデータを使用する．

コード 5: ブロードキャスト

```
json = '{"dpid": ' + Datapath_ID + ',"cookie
    ": 0,"table_id": 0,"priority": 10,"flags
    ": 1,"match":{"eth_dst": "ff:ff:ff:ff:ff
    :ff"},"actions":[{"type":"OUTPUT","port
    ": "OFPP_FLOOD"}]}'
r = requests.post(url,json)
```

5.3.7 フローエントリの追加・確認機能

コード 6 にスイッチに設定されているフローエントリの確認およびフローエントリの追加処理を示す．スイッチの識別子である Datapath_ID を指定して URL にアクセスすることで，該当するスイッチのフローエントリが返答される．また，エントリの追加については，5.3.4

```
~~~ Connected to database! ~~~
----------------------
Check flow table     :1
Initialize flow table:2
Add flow entry       :3
Enter number    →3
----------------------
Start broadcasting:1
Remove Action     :2
Add flow entry    :3
Enter number    →3
----------------------
Enter the name of the switch to configure:router
Node_ID: 1
Node_name: router
Datapath_ID: 1
Port: 8080
IP_Address: 192.168.45.2
----------------------
Generated URL
http://192.168.45.2:8080/stats/flowentry/add
----------------------
Enter the host name to add the flow entry:host1
Node_ID: 2
Node_name: host1
Datapath_ID: 1
Priority: 10
Source_MAC_Address: 84:47:09:11:64:92
Source_Port: 5
----------------------
JSON output
{"dpid": 1,"priority": 10,"match":{"eth_dst":"84:47:09:11
:64:92"},"actions":[{"type":"OUTPUT","port":5}]}
----------------------
<Response [200]>
----------------------
END
```

図 3: host1 フローエントリーの追加

と 5.3.5 で示した URL と JSON 形式のデータを使用することで，指定したスイッチにエントリを追加できる．

コード 6: フローエントリの確認・追加

```
1   // 確認用 URL
2   url = 'http://' + IP_Address + ':' + Port
        + '/stats/flow/' + Datapath_ID
3   r = requests.get(url)
4   // 追加用 URL
5   r = requests.post(url,json)
```

6 動作検証

作成した Engine プログラムを使用して，OpenFlow スイッチの router に接続されている host1 と host2 間で通信できるように設定変更ができるかを検証した．図 3 は本稿で実装した Engine プログラムを動作させたときの様子である．スイッチに対して host1 が接続されていることを示すフローエントリを設定している．5 節で示したとおり，REST API の URL を作成し，設定に必要な JSON 形式のデータを生成，スイッチへ送信している．図 3 下部の <Response [200]> が表示されれば設定が完了したことを示している．同様の手順で host2 の設定も実施した．さらに，5.3.6 で概説したブロードキャストパケットを処理するためのエントリの設定結果を図 4 に示す．最後に，スイッチに設定されているフローエントリを図 5 に示す．二台の端末が接続されており，ブロードキャストパケットについてはフラッディングするように設定されていることから，一般的なスイッチングハブの機能を実現してることになる．ping コマンドによって端末間のネットワーク疎通性を確認することができたことから，想定した設定や動作していることがわかった．

```
----------------------
Generated URL
http://192.168.45.2:8080/stats/flowentry/add
----------------------
JSON output
{"dpid": 1,"cookie": 0,"table_id": 0,"priority": 10,"flag
s": 1,"match":{"eth_dst": "ff:ff:ff:ff:ff:ff"},"actions":
[{"type":"OUTPUT","port": "OFPP_FLOOD"}]}
----------------------
<Response [200]>
----------------------
END
```

図 4: ブロードキャストの設定

```
Generated URL
http://192.168.45.2:8080/stats/flow/1
----------------------
Result
{"1": [{"priority": 10, "cookie": 0, "idle_timeout": 0, "
hard_timeout": 0, "byte_count": 0, "duration_sec": 864, "
duration_nsec": 210000000, "packet_count": 0, "length": 8
8, "flags": 0, "actions": ["OUTPUT:5"], "match": {"dl_dst
": "84:47:09:11:64:92"}, "table_id": 0}, {"priority": 10,
 "cookie": 0, "idle_timeout": 0, "hard_timeout": 0, "byte
_count": 0, "duration_sec": 232, "duration_nsec": 2640000
00, "packet_count": 0, "length": 88, "flags": 0, "actions
": ["OUTPUT:6"], "match": {"dl_dst": "00:1b:21:ea:c4:47"}
, "table_id": 0}, {"priority": 10, "cookie": 0, "idle_tim
eout": 0, "hard_timeout": 0, "byte_count": 0, "duration_s
ec": 219, "duration_nsec": 483000000, "packet_count": 0,
"length": 88, "flags": 1, "actions": ["OUTPUT:FLOOD"], "m
atch": {"dl_dst": "ff:ff:ff:ff:ff:ff"}, "table_id": 0}]}
----------------------
END
```

図 5: フローテーブルの確認

7 おわりに

本稿では，Ryu コントローラの設定方法を調査し，Ryu REST API の構成法について検討した．物理ネットワークの結線状態や OpenFlow スイッチの状態，管理者からの設定内容などを格納するデータベースを設計・実装した．そのデータベースの情報をもとに，ネットワーク設定を HTTP 経由で送信することができる REST API を生成する Engine プログラム実装し，動作検証した．今後の課題は，より複雑なネットワーク制御を考慮したエンジンの設計や異なるコントローラに対しても設定できるようなドライバの設計を進める必要がある．

謝辞

本研究は JSPS 科研費 JP20K11802 の助成を受けたものです．

参考文献

[1] Imae, A., Mino, K., Koyama, O., Oyama, K., Yamaguchi, M., Ikeda, K. and Yamada, M.: Router Control Function Using IoT Device Supported OpenFlow Switch in IP over AWG-STAR Network, *Proceedings of OECC*, Taipei, Taiwan, pp. 1–3 (2020).

[2] Tran, H. M., Le, S. T., Nguyen, S. V. and Le, H.-D.: A Web-Based Management System for Software Defined Network Controllers, *Proceedings of ACOMP*, Nha Trang, Vietnam, pp. 1–6 (2019).

[3] 佐藤 寧洋, 河合 勇輝, 阿多 信吾, 岡 育生: データベースを利用した SDN アーキテクチャの動的構成手法, **電子情報通信学会論文誌**, Vol. J100-B, No. 12 (2017).

[4] Ryu SDN Framework Community: Ryu SDN Framework. https://ryu-sdn.org.

[5] ORACLE: MySQL Connector/Python Developer Guide. https://dev.mysql.com/doc/connector-python/en/.

[6] Reitz, K.: Requests: HTTP for Humans. https://docs.python-requests.org/en/latest/.