Vol.2015-IOT-31 No.13
Vol.2015-SPT-15 No.13
2015/9/26

Netspec: OpenFlowネットワークのテスト基盤の提案と試作

原 一貴[1,a)]　渡場 康弘[1]　市川 昊平[1]　飯田 元[1]

概要：近年，SDN およびそれを実現する OpenFlow により，ネットワークをプログラマブルに制御可能な環境が整いつつある．このようなネットワークの動作が，開発者の設計どおりであることを保証するためには，ネットワークを制御する OpenFlow コントローラのテストが重要であり，OpenFlow コントローラの単体テストの手法が一般的なプログラム開発で行われる単体テストと同様に確立されつつある．しかし，OpenFlow コントローラの挙動は，配備するネットワークの構成にも依存するため，OpenFlow コントローラの単体テストだけでは不十分である．そのため，実際に配備しようとするネットワークと OpenFlow コントローラを結合させた時の挙動をテスト可能とする必要があるが，現状ではそれを実現する手法は十分に確立されているとは言えない．本稿では，ネットワーク全体が要求仕様どおりに動作するか検証するテスト基盤 Netspec を提案し，その試作と検証結果に関して報告する．

1. はじめに

近年，新しいネットワークの制御手法として，Software-Defined Networking（SDN）が普及しつつある [1]．従来のネットワークにおけるネットワークスイッチは，パケットの転送の方針を決定・管理する制御機能と，転送の方針にしたがって実際にパケットの転送を行う転送機能から構成される．ネットワーク全体の挙動は各スイッチに実装されている制御機能が互いに連携しパケットの転送機能を制御することで決定される．ネットワーク管理者は分散する各スイッチの制御機能に対して転送方針の矛盾が無いように設定する必要がある．設定作業はネットワークスイッチごとに検討して行う必要が有るため，手作業によるところが大きく煩雑である．しかし，近年に導入が進んでいる仮想化技術を用いたクラウドサービスによって，ネットワークに対する要求は動的に変化し続けるようになり，これらの要求に対し柔軟に対応する必要が生じている．そこで人手に頼ったネットワーク管理に代わり，ソフトウェアによって一元的かつ柔軟にホスト間の通信を一つのフローとして制御する SDN が注目されるようになった．SDN では各ネットワークスイッチに分散する制御機能をネットワークスイッチから分離し，制御機能をコントローラに集約する．コントローラはソフトウェアとして構築できるため，ネットワークの挙動をプログラマブルに制御することができる．

SDN を実現する技術の一つとして OpenFlow [2] がある．OpenFlow におけるネットワークは，制御機能を担当する OpenFlow コントローラとパケット転送処理を行う OpenFlow スイッチで構成される．OpenFlow コントローラと OpenFlow スイッチの間は，SecureChannel と呼ばれる通信路で接続されており，OpenFlow プロトコルに基づきネットワークの制御が行われる．

しかし，ソフトウェアである OpenFlow コントローラは，一般的なソフトウェアと同じくバグを含む可能性がある．OpenFlow ネットワークにおいて OpenFlow コントローラのバグは，ネットワークの挙動に対して直接的に影響を及ぼすため，ネットワークを利用する様々なアプリケーションに対し広く影響を与えることになる．そのため，OpenFlow ネットワークの挙動をテスト可能とする仕組みが重要だが，十分に確立されているとは言えない [3], [4], [5], [6]．そこで，われわれは OpenFlow コントローラを実際に配備するネットワークを想定した検証を可能とするテスト基盤の実現を目指す．本稿ではこのテスト基盤で必要となる機能要件について分析を行い，その機能を実装したプロトタイプ実装に関してその有用性を評価する．

本稿の構成は以下のとおりである．2 節では OpenFlow コントローラの開発におけるテストに関する研究について述べる．3 節では提案する OpenFlow ネットワークのテスト基盤を実現するための機能要件について分析を行う．4

[1] 奈良先端科学技術大学院大学　情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology
[a)] harakazuki.he7@is.naist.jp

節では提案システムの構成及び実装について説明する．5節では提案システムのプロトタイプを用いて各機能の評価を行い，6節では提案するテスト基盤の有用性について議論する．7節ではまとめと今後の研究について述べる．

2. 関連研究

OpenFlow コントローラのバグはネットワークを利用するアプリケーションに対し広く影響を及ぼすため，バグへの対策としてさまざまな研究が取り組まれている．本節では，本稿に関連する研究とツールについて述べる．

2.1 ndb

Nikhil らは，ネットワーク上の任意のフローがどのスイッチを経由したか追跡可能にすることによって，OpenFlow コントローラにおけるデバッグ作業を支援するツールである ndb を提案した [5]．ndb では，SecureChannel 上でメッセージを中継することで，OpenFlow コントローラが OpenFlow スイッチに対して送ったフロー制御情報を監視する．そして，特定のフローに対してそのコピーをフロー収集ホストが接続されたポートに転送するように制御情報を改変し，収集したフローのコピーを元に ndb はフローが通過した経路を開発者に提示する．OpenFlow コントローラとスイッチの間で OpenFlow メッセージを中継する手法を用いることで，ndb は OpenFlow コントローラの変更を不要にする．しかし，OpenFlow スイッチに対してフロー収集ホストを接続してフロー制御情報を改変するため，デバッグ専用のネットワーク環境が必要であり，実際に配備する OpenFlow ネットワークに対して ndb を用いることが難しい．

2.2 NICE

SecureChannel におけるメッセージの到達タイミングにより起因する OpenFlow コントローラの想定外の動作を検証可能とするツールとして Marco らは NICE を提案した [3]．NICE では，OpenFlow コントローラ，OpenFlow スイッチ，ホストをモデル化し，モデル間の相互動作を OpenFlow コントローラのイベントハンドラと共に記号実行することで生成された状態空間からバグの検出を行う．NICE はモデルベースの手法であるため，改変を行うことなく OpenFlow コントローラの検証が可能である．しかし，NICE は OpenFlow スイッチをモデル化して扱うため，実際に配備するネットワークの特性を反映した検証が実施できない．

2.3 OFTEN

Maciej らは，前項の NICE を元にスイッチモデルに対して OpenFlow スイッチを割り当てる手法を用いた OpenFlow コントローラの検証手法 OFTEN を提案している [6]．NICE で用いられていたスイッチのモデルに対して実際のスイッチ実装を割り当てることにより，NICE [3] では実施できなかったスイッチが持つ特性を反映した OpenFlow コントローラの検証が可能となる．OFTEN は検証用のフローをネットワーク上に送信する必要が有るため，ネットワークを構成するすべての OpenFlow スイッチと OFTEN が動作するホストを接続する必要がある．それゆえ，OFTEN ではネットワークの配線を検証用に変更することとなるため，既存の OpenFlow ネットワークへの適用が困難である．

2.4 Mininet

Mininet [4] は，OpenFlow コントローラの研究開発における動作検証を行うツールとして広く用いられている．Mininet は，仮想的に OpenFlow ネットワークとホストを用いた検証基盤を構築する．開発者は自由に仮想ネットワークを構築することにより，OpenFlow コントローラの振る舞いを自由に検証できる．しかし，Mininet は OpenFlow コントローラの動作検証を独自の仮想ネットワーク上で実現するため，OpenFlow コントローラを実際に配備するネットワーク環境の特性を考慮した検証ができない．

以上のように，OpenFlow コントローラ単体を検証するための手法が研究開発されているが，実際の OpenFlow ネットワークの特性を反映した検証基盤は十分に確立されているとは言えない．

3. OpenFlow ネットワークの検証・テストにおける問題

前節までに述べたおり OpenFlow コントローラに対する単体テストやその動作の検証手法は，OpenFlow コントローラの品質を保つ上で一定の効果があると考えられる．しかし，OpenFlow ネットワーク全体の挙動を保証するためには，OpenFlow コントローラにのみ注目した検証のみでは不十分であると考える．以下，考えられる問題に関して整理する．

OpenFlow ネットワークの構成要素は OpenFlow コントローラとネットワークを構成する OpenFlow スイッチからなり，OpenFlow コントローラの単体テストだけではネットワーク全体の挙動の検証としては不十分であると考えられる．OpenFlow コントローラの単体テストの場合は，ソフトウェアで構成された仮想スイッチを用いることで実施するが，実際に配備するネットワークで用いるスイッチと検証用の仮想スイッチは基本的に構成が異なる．そのため，単体テスト過程では現れない不具合が実際のネットワーク上で出現する可能性がある．

また，OpenFlow コントローラにネットワークのトポロジに起因する不具合が存在する場合，単体テストなど OpenFlow コントローラ自身の検証だけでは不十分であると考えられる．実際に OpenFlow コントローラを配備しよ

うとするネットワークにおいて，ネットワークが期待するとおりに動作することを保証するためには，実際のトポロジと同一かまたは同等の環境において検証可能とすることが重要である．

以上より，OpenFlow ネットワークのテスト基盤は，ネットワークに対する要件が満たされているか検証可能である必要があると考える．そこでわれわれは，OpenFlow ネットワークの挙動の検証におけるこれらの問題を解決するために，OpenFlow ネットワーク全体を対象としたテスト基盤を提案する．OpenFlow ネットワーク全体をテスト基盤の対象とすることで，OpenFlow コントローラを配備しようとするネットワークのトポロジやネットワークの構成をふまえた検証の実現を目指す．

さらに，OpenFlow ネットワーク全体を対象としたテスト基盤に必要となる機能要件について以下のように整理する．

要件 1　テスト基盤は OpenFlow コントローラや OpenFlow スイッチに対し非侵襲的な手法を用いて検証に必要な情報を取得できる．

要件 2　テスト基盤は要件 1 で取得した情報を管理できる．

要件 3　テスト基盤は OpenFlow コントローラを配備するネットワークを用いて検証を実施できる．

要件 4　開発者は任意の検証用プログラムを作成し，それを用いてネットワークの検証を実施できる．

4. OpenFlow テスト基盤の設計と実装

本研究では，前節で検討した機能要件を満たす OpenFlow ネットワーク全体を対象としたテスト基盤のプロトタイプ，Netspec を提案する．Netspec はサーバ設定の検証において広く採用されている serverspec [7] の概念に基づいて提案したツールである．

サーバ設定の分野では Chef [8]，Puppet [9]，Ansible [10] などサーバ設定を自動化する構成管理ツールが普及しつつある．しかし，これらの構成管理ツールによって自動設定された実際のサーバシステムが，仕様どおりに設定がなされたことを確認する手段が必要である．serverspec [7] は自動設定されたサーバシステムを実際に運用する前に，設定が仕様どおりに完了しているかを検証するツールである．Netspec はその仕組みを参考にネットワークを対象としたツールを提案するものである．OpenFlow コントローラによって設定されたネットワークが，仕様どおりに挙動するか実際にントローラプログラムが配備されるネットワークを用いて検証できる環境を提供することが狙いである．

4.1 システム構成

Netspec は，OpenFlow コントローラとスイッチの間で OpenFlow メッセージを中継し情報を収集する Netspec Proxy Modules と，検証全体を管理する Netspec Manager，

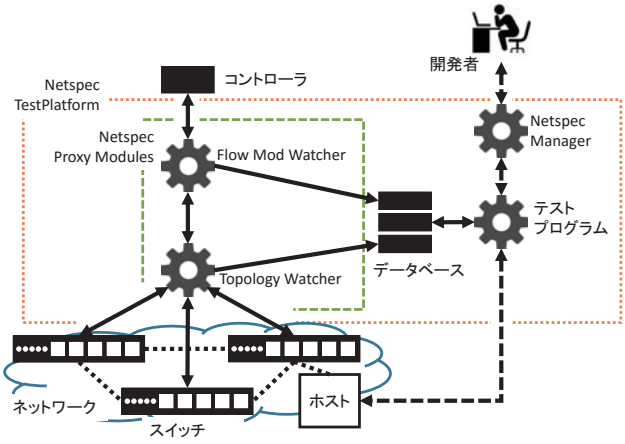

図 1　Netspec の概要図

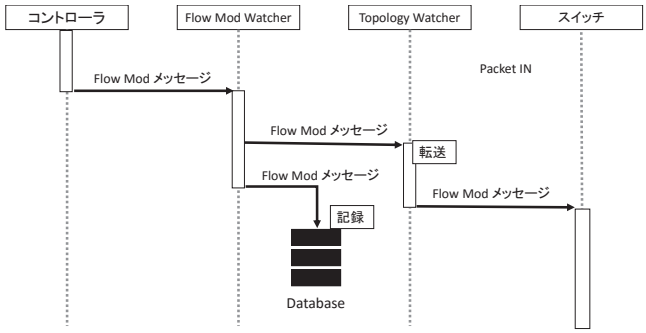

図 2　Flow Mod Watcher の動作フロー

Netspec Proxy Modules によって取得された情報を保管するデータベース，検証要件に基づいてデータベースに記録された情報を元に分析を行うテストプログラムで構成される．また，Netspec Proxy Modules は Flow Mod メッセージを監視・記録する Flow Mod Watcher とトポロジを把握し記録する Topology Watcher から成る．

4.1.1 Netspec Manager

Netspec Manager は，検証全体の流れを管理し，開発者に対して OpenFlow ネットワークが要求仕様どおりの動作を実行しているか，テスト結果の成否を通知する．Netspec Manager は 3 節の要件 3 の機能，および要件 1 と要件 2 を管理する機能を有する．

4.1.2 Flow Mod Watcher

Flow Mod Watcher は，OpenFlow コントローラとネットワークスイッチの間のプロキシプログラムとして動作する．プロキシプログラムとして動作する仕組みは，ndb と同様であり，OpenFlow コントローラとスイッチの間で OpenFlow メッセージを中継する．OpenFlow コントローラやスイッチに修正を加えることなく，非侵襲的に Flow Table や OpenFlow コントローラからの Flow Mod メッセージを収集可能である．3 節の要件 1 の機能は，Flow Mod Watcher により提供される．

Flow Mod Watcher の動作フローを**図 2** に示す．Flow

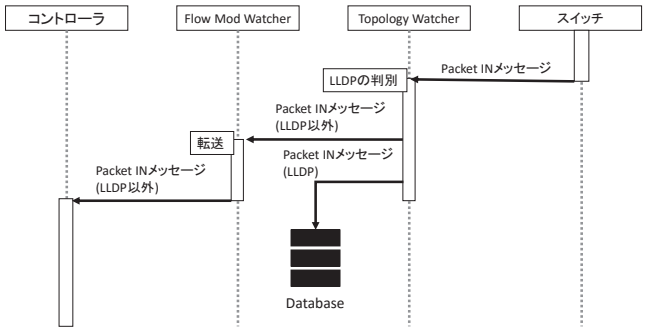

図 3 Topology Watcher の動作フロー

Mod Watcher はスイッチ側からの接続を受け取ると，OpenFlow コントローラとネットワークスイッチの接続を管理するプロセスを立ち上げる．このプロセスは OpenFlow コントローラ側に対して接続要求を行い，OpenFlow コントローラ側の接続を確立する．その後，OpenFlow コントローラ側とネットワークスイッチ側の通信を監視し，Flow Mod メッセージを受け取ると，そのメッセージをデータベースへ保存する．

4.1.3 Topology Watcher

Topology Watcher は，Flow Mod Watcher と同じく，OpenFlow コントローラと OpenFlow スイッチの間のプロキシプログラムとして動作することで，OpenFlow コントローラやスイッチに修正を加えることなく，非侵襲的に，ネットワークのトポロジを把握可能である．Topology Watcher により，3 節の要件 1 の機能が提供される．

Topology Watcher の動作フローを**図 3** に示す．Topology Watcher もまた，スイッチ側より新たに接続を受け取ると，OpenFlow コントローラとスイッチの接続を管理するプロセスを立ち上げる．Topology Watcher は OpenFlow コントローラとスイッチの接続を確立した後，LLDP パケットをスイッチ側に Packet Out メッセージを使って送信する．LLDP パケットにはスイッチの識別子と送信元ポート番号が含まれ，各スイッチのポートで近接のスイッチに送信される．

LLDP パケットを受け取ったスイッチは，Packet In メッセージを使って Topology Watcher に LLDP パケットを転送する．メッセージを受け取った Topology Watcher は，Packet In メッセージを送ったスイッチ（LLDP パケットの受信側スイッチ）の識別子と，Packet In メッセージに含まれる受信ポート（LLDP パケットを受け取ったポート），LLDP パケットに含まれる送信元ポートと送信元スイッチの識別子をデータベースに書き込む．これによって，Topology Watcher はネットワーク全体のトポロジを把握する．

4.1.4 データベース

データベースは，Flow Mod Watcher で取得された OpenFlow メッセージ，および Topology Watcher で取得したトポロジ情報を保存する．本プロトタイプでは MongoDB を用いて実装し，OpenFlow コントローラの検証のためにテストプログラムから情報にアクセス可能としている．データベースは 3 節の要件 2 の機能を提供する．

4.1.5 テストプログラム

開発者は，検証要件に合わせて，データベースに記録された OpenFlow メッセージを分析し，ネットワークが要求仕様どおりの動作を実行しているか確認するテストプログラムを作成する．例えば，ネットワーク上の 2 つホスト間におけるパケットの到達性を検証したいとした場合のテストプログラムの例を**図 4** に示す．このテストプログラムでは，送信元ホストと接続されているスイッチの入力ポートから，受信先ホストが接続されているスイッチのポートまでの通信フローの経路を追跡することによって到達性の検証を実現している．開発者は，このようなテストプログラムを使い，データベースに記録された OpenFlow メッセージを分析することにより，ネットワークが要求仕様どおりの動作を実行しているか，テスト結果の成否を確認することができる．3 節の要件 4 の機能は，テストプログラムにより提供される．

4.2 Netspec を用いた OpenFlow ネットワークテストの流れ

実際のテストにおいては，開発者は次の流れで Netspec を用いて OpenFlow ネットワーク全体が要求仕様どおりの動作をするか検証を行う．

(1) 開発者は検証要件を元に，OpenFlow メッセージテストプログラムを作成する．
(2) 開発者は検証に必要な，OpenFlow ネットワークに接続されたホストから送信する検証に必要なパケットを準備する．
(3) 開発者は Netspec Manager を起動する．
(4) Netspec Manager は OpenFlow コントローラ，Flow Mod Watcher，Topology Watcher の順に必要なモジュールを起動する．
(5) Topology Watcher はトポロジの監視を行う．
(6) Netspec Manager は OpenFlow ネットワーク上のホストにアクセスし，検証に必要なパケットをホストからネットワークに送信する．
(7) Flow Mod Watcher はネットワークに送信されたパケットによって起因する OpenFlow コントローラからの Flow Mod メッセージを監視・記録する．
(8) Netspec Manager は，(6) および (7) が終了すると，テストプログラムを実行し，ネットワークが要求仕様どおりの動作を実行しているかテスト結果の成否を開発者に通知する．

現状のプロトタイプ実装では，上記の (6) および (8) の動作はまだ自動化されておらず，本報告時点では手作業によ

```
def check_reachability(スイッチ，宛先 MAC アドレス，受信ポート番号，トポロジ)
  matched_rules =
    スイッチのルールから宛先 MAC アドレスと受信ポート番号を対象とするフローエントリーを抽出
  if matched_rules.empty?
    return False
  rule = matched_rules[0]
  if rule に対応する次のスイッチが存在しない
    return True
  else
    return check_reachability(rule に対応する次のスイッチ，
      宛先 MAC アドレス，次のスイッチの入力ポート番号，トポロジ)
```

図 4 到達性確認のアルゴリズム

り実行し評価した．

5. 評価実験

本論文で提案したプロトタイプが，3 節で述べた OpenFlow ネットワーク全体を対象としたテスト基盤の機能要件を満たすか次の実験を行った．

5.1 実験 1

目的 Flow Mod Watcher と Topology Watcher による SecureChannel に対する影響の計測

対応する要件項目 要件 1

手法 OpenFlow コントローラと Flow Mod Watcher，Flow Mod Watcher と Topology Watcher，Topology Watcher とネットワークスイッチのそれぞれの間でメッセージの通過時刻を計測・比較し，Flow Mod Watcher と Topology Watcher がそれぞれのメッセージの転送にかかる時間を算出する．

5.2 実験 2

目的 Netspec を用いたネットワークフローの到達性の確認を実現

対応する要件項目 要件 2，要件 3，要件 4

手法 Mininet [4] を用いて**図 5** の実験用のネットワーク環境を構築する．そして，検証用のパケットフローを発生させるためにホスト 1 からホスト 2 に対して ping コマンドを実行し，それに起因する OpenFlow コントローラからの Flow Mod メッセージとトポロジ情報をデータベースに記録する．さらに，これらの情報に到達性を確認する図 4 のテストプログラムを実行し，到達性が確認できるか検証する．

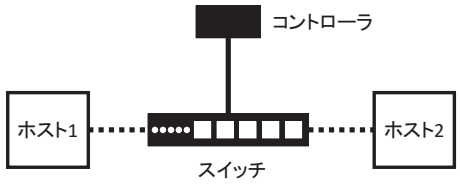

図 5 評価実験用トポロジ

6. 結果と考察

6.1 実験 1

実験 1 の結果を**表 1**，**表 2** に示す．表 1，表 2 は，ARP Request，ARP Reply，ICMP Echo Request の各パケットに対して，OpenFlow コントローラおよびスイッチから送信される Packet In，Flow Mod，Packet Out メッセージが Flow Mod Watcher および Topology Watcher を通過する際に掛かる遅延時間を示している．この結果より，各 OpenFlow メッセージが Flow Mod Watcher および Topology Watcher を通過する際にかかるオーバヘッドは，多くの場合 1ms 以下であり，最高でも 2ms 以下であることが確認できた．

6.2 実験 2

Nespec の検証にあたり，4 節で示した図 4 の到達性を確認するテストプログラムを用いてデータベースに記録されている OpenFlow メッセージとトポロジ情報を分析したところ，ホスト 1 からホスト 2 への到達性の確認ができた．到達性を確認するテストプログラムは次の流れで動作し到達性の検証を行う．

(1) テストプログラムは，データベースに記録されている Flow Mod メッセージを元に，スイッチごとの Flow Table を構築する．
(2) テストプログラムは，データベースに記録されているトポロジ情報を取得する．

表 1 実験 1 の結果（Flow Mod Watcher）単位：ミリ秒

	ARP Req.	ARP Reply	ICMP Echo Req.
PACKET IN	0.154	0.093	0.337
FLOW MOD	—	1.889	0.951
PACKET OUT	0.291	1.894	0.956

表 2 実験 1 の結果（Topology Watcher）単位：ミリ秒

	ARP Req.	ARP Reply	ICMP Echo Req.
PACKET IN	0.762	0.702	0.59
FLOW MOD	—	0.19	0.437
PACKET OUT	0.09	0.19	0.462

(3) テストプログラムは，ホスト 1 が接続されているスイッチのポートからパケットの入力があったと仮定し，そのスイッチの Flow Table を参照してパケットの転送先を導き出す．
(4) データベースにあるトポロジ情報を参照し，次のスイッチを見つける．
(5) 今回の場合，次のスイッチが見つからないので宛先ホストに到達したと判断し，到達可能であることを開発者に通知する．
(6) もし，次のスイッチが見つかった場合は (3) に戻りくりかえす．また，途中で転送ルールが存在しない場合は到達が不可能であることを開発者に通知する．

これにより，OpenFlow メッセージとトポロジ情報の分析によってネットワークの検証が実現できること確認した．

6.3 考察

SecureChannel の通信に対して，データベースへのメッセージ書き込みによるオーバヘッドが確認されるが，1〜2ms 程度のオーバヘッドであるため OpenFlow ネットワーク全体に影響は少ないと考えられる．また，収集された情報を分析することによってネットワークテストを実現しているため，OpenFlow ネットワークに対する影響は発生しない．したがって，本プロトタイプの手法は実用的であると考えられる．

7. まとめと今後の課題

本稿では，OpenFlow ネットワーク全体の挙動に対し検証を実行可能とする Netspec を提案した．また Netspec が，3 節で述べた OpenFlow ネットワーク全体を対象としたテスト基盤に対する機能要求を満たしているか検証を行った．

今後は，テストプログラムの作成を容易にするため，ドメイン特化言語を用いたテストプログラム開発フレームワークに関する研究開発を行う．また，本稿は実装したプロトタイプの検証のためにネットワークの到達性確認のテストプログラムを用いたが，他に検証すべき項目や手法について研究を行う．本プロトタイプでは，検証用フローの生成を OpenFlow ネットワークに接続されたホストから手作業によって行ったが，テストの自動化に向けて検証用フローの自動生成機能に関する研究を行う．

謝辞 本研究の成果の一部は科研費 (15K00170) の助成を受けたものである．

参考文献

[1] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, “A Survey on Software-Defined Networking”, IEEE Communications Surveys and Tutorials, vol. 17, no. 1, pp. 27–51, 2015.
[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: Enabling Innovation in Campus Networks”, SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69–74, 2008.
[3] M. Canini, D. Venzano, P. Perešíni, D. Kostić, and J. Rexford, “A NICE Way to Test OpenFlow Applications”, The 9th USENIX conference on Networked Systems Design and Implementation(NSDI’12), pp. 127–140, 2012
[4] “Mininet: Rapid prototyping for software defined networks”, http://mininet.org/ .
[5] N. Handigol, B. Heller, V. Jeyakumar, D. Maziéres, and N. McKeown, “Where is the debugger for my software-defined network?”, The 1st workshop on Hot topics in software defined networks (HotSDN ’12), pp. 55–60, 2012.
[6] M. Kuzniar, M. Canini, and D. Kostic, “OFTEN Testing OpenFlow Networks”, 2012 European Workshop on Software Defined Networking (EWSDN), pp. 54–60, 2012.
[7] 宮下 剛輔, 栗林 健太郎, 松本 亮介, “serverspec: 宣言的記述でサーバの状態をテスト可能な汎用性の高いテストフレームワーク”, 情報処理学会研究報告 2014-IOT-24(15), pp.1–6, 2014.
[8] “Chef”, https://www.chef.io/chef/ .
[9] “Puppet Labs”, https://puppetlabs.com/ .
[10] “Ansible”, http://www.ansible.com/ .