レイヤ7情報に基づいたサーバ選択を可能にするOpenFlowの機能拡張

五十嵐 裕也[1,†1]　山井 成良[2,a)]　北川 直哉[2]

概要：
OpenFlow に代表される SDN（Software Defined Network）の出現により，たとえばサーバ負荷分散やトラフィック分散など，柔軟なトラフィック制御が可能になっている．しかし，SDN では原則としてレイヤ 4 以下の情報に基づいて処理内容を決定しているため，たとえばコマンドの引数などレイヤ 7 の情報に応じたサーバ選択が行えない．そこで，本論文ではこのような柔軟なトラフィック制御を可能にするために必要な，OpenFlow における TCP コネクションの確立後のホスト変更機能を提案する．この機能により，クライアントからのコネクション確立要求を受け取ると 1 つのサーバとの間で一旦コネクションを確立しながら通信内容を監視し，通信開始後にその内容に応じてサーバを切り替えることができるようになる．

キーワード：
OpenFlow，SDN，レイヤ 7 スイッチ

Enhancement of OpenFlow Functions for Server Selection Based on Layer 7 Information

Hiroya Ikarashi[1,†1]　Nariyoshi Yamai[2,a)]　Naoya Kitagawa[2]

Abstract:
With the advent of Software Defined Network (SDN) such as OpenFlow, flexible traffic control such as load balancing amang servers, traffic distribution, and so on can be performed. However, since SDN basically decides how to process incoming packets based on information of layers 4 or less, server selection according to layer 7 information such as command argument cannot be performed. In this paper, to enable such flexible traffic control, we propose a host altering function required after TCP connection establishment in OpenFlow. With this function, on receiving the connection establishment request from the client, it is possible to monitor the payload of communication after establishing a connection with one server, and alter the server according to the payload.

Keywords:
OpenFlow，SDN，Layer 7 switch

1. はじめに

通信端末の増加やネットワーク機能の多様化に伴って，ネットワークの構造やその制御は複雑化・高度化している．これに対して，コンピュータの性能が向上することで，ソフトウェアを用いてネットワークを柔軟かつ動的に制御する技術が研究されている．その技術の概念の 1 つが SDN（Software Defined Network）である．代表的な SDN とし

[1] 東京農工大学大学院工学府
Graduate School of Engineering, Tokyo University of Agriculture and Technology
2-24-16, Nakacho, Koganei, Tokyo 184–8588, Japan
[2] 東京農工大学工学研究院
Institute of Engineering, Tokyo University of Agriculture and Technology
2-24-16, Nakacho, Koganei, Tokyo 184–8588, Japan
[†1] 現在，株式会社日立システムズ
[a)] nyamai@cc.tuat.ac.jp

て OpenFlow[1] がある．OpenFlow では，OSI 参照モデルのレイヤ 1 からレイヤ 4 までのヘッダフィールドのうち，定められた 40 以上の項目に基づいたパケットの制御が可能である [2]．

OpenFlow を用いた既存のトラフィック制御の研究として，サーバや通信路の負荷分散，不正アクセスや DDoS(Distributed Denial of Service) 攻撃の検知と排除によるセキュリティ対策 [3], [4], [5] などを目的とするものが挙げられる．しかし，これらの研究のほとんどはレイヤ 4 以下の情報に基づいて処理内容を決定しているため，たとえばレイヤ 7 での通信内容に基づいて経路を決定することができない．

これに対して，REFLO[6] では内部ネットワークと外部ネットワークの境界に複数のファイアウォールが設置されている環境において，外部のクライアントと内部のサーバとの間でまず TCP コネクションを確立させ，その後に通信内容に基づいて経路切替を行い，通過させるファイアウォールを選択する方法が提案されている．この方法ではファイアウォールの選択後にそれまで行われた通信内容を選択したファイアウォール上で再現することにより，ステートフルな検査を行うことが可能である．しかし，この方法では通信内容に基づいたファイアウォールの選択は可能であるものの，通信内容に基づいたホスト（サーバ）の選択は行えない．

そこで，本論文では通信内容に応じた柔軟なトラフィック制御を可能にするために必要な，OpenFlow における TCP コネクションの確立後のホスト変更機能を提案する．この機能により，クライアントからのコネクション確立要求を受け取ると 1 つのサーバとの間で一旦コネクションを確立しながら通信内容を監視し，通信開始後にその内容に応じてサーバを切り替えることができるようになる．

以下，2 節では本研究の元となった関連研究について述べる．次に 3 節では TCP コネクションの確立後のホスト変更機能を実現するために必要な OpenFlow スイッチの機能拡張について述べ，4 節ではその実装方法について説明する．引き続き，5 節では試作システムの性能評価について述べる．

2. 関連研究

前節で述べたように，OpenFlow では通常はレイヤ 7 での通信内容に基づいて処理を行うことができないが，REFLO[6] ではレイヤ 7 での通信内容に基づいてファイアウォールを選択することができる．本節では関連研究として REFLO の構成および動作について述べる．

2.1 REFLO の構成と動作

REFLO では図 1 に示すようにクライアントとサーバとの間に役割の異なる複数のファイアウォール（直接接続リンクを含む）[*1] を配置し，2 基の OpenFlow スイッチで挟む構成を取る．この構成においてクライアントの IP アドレスやポート番号などのレイヤ 4 以下の情報に基づいてクライアント・サーバ間の各フローが通過するファイアウォールを決定する場合には，当該フローの最初のパケット（TCP の場合には SYN パケット）が OpenFlow スイッチに到着するとそのパケットは Packet In メッセージにより OpenFlow コントローラに送られ，OpenFlow コントローラはレイヤ 4 以下の情報に基づいて当該フローの往復のパケットが特定のファイアウォールを通過するように 2 基の OpenFlow スイッチに Flow Mod メッセージを用いてフローエントリを登録する．その後，OpenFlow コントローラは Packet In メッセージを送った OpenFlow スイッチに Packet Out メッセージを送り，通信を継続する．

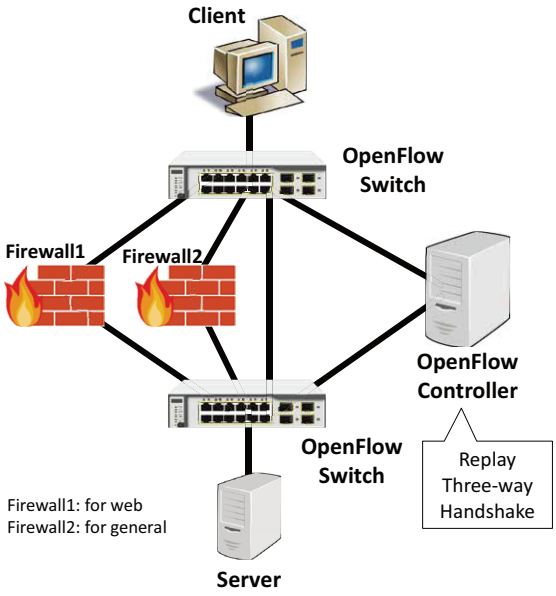

図 1　REFLO のシステム構成

一方，クライアント・サーバ間の各フローが通過するファイアウォールをレイヤ 7 情報に基づいて決定する場合には，決定に必要な情報が得られるまでの間は OpenFlow コントローラはフローエントリを 2 基の OpenFlow スイッチに登録せず，直接接続リンクを経由するように OpenFlow スイッチを制御する．特に当該フローが TCP の場合，3 ウェイハンドシェークのパケットはレイヤ 7 情報を含まないため，全て Packet In メッセージにより OpenFlow コントローラに送られる．OpenFlow コントローラはこれらのパケットを後に利用するために記録しておく．次に，OpenFlow コントローラが決定に必要な情報を含むパケットを受け取ると，OpenFlow コントローラは初めてフローエントリを 2 基の OpenFlow スイッチに登録し，このパケットを含むこれ以降のパケットを OpenFlow コントローラの介在なしで特定のファイアウォールを経由するように

[*1] 図 1 では Firewall1 が Web 専用，Firewall2 が一般用と想定している．

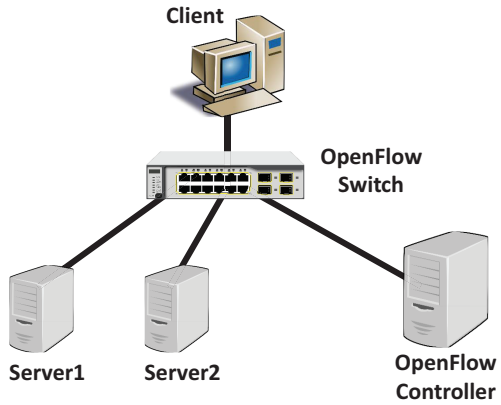

図 2　想定するネットワーク構成

する．しかし，これだけではファイアウォールから見るとたとえば 3 ウェイハンドシェークが行われないまま TCP 通信が行われているように見えるため，誤動作が発生する危険性がある．そこで REFLO では 2 基の OpenFlow スイッチ間でこれまでに記録した当該フローの先行パケットを選択したファイアウォールを経由するように再現し，その後上記のフローエントリ登録処理を行うようにする．

2.2　REFLO の問題点

REFLO ではレイヤ 7 情報に基づいてクライアント・サーバ間の経路を決定することが可能であり，この機能を用いて各フローが通過するファイアウォールを選択することができる．しかし，複数のサーバが設置されている環境において TCP 通信を行う際，レイヤ 7 スイッチのようにレイヤ 7 情報に基づいてサーバを選択したい場合に REFLO ではこれを行えない点が問題となる．たとえ複数のサーバ間で同一の IP アドレスを共有したとしても，各サーバの ISN（Initial Sequence Number）が異なるためサーバ間で TCP 通信の引継ぎができず，この問題を解決できない．

3.　TCP 通信のサーバ間引継ぎを可能とする OpenFlow の機能拡張

REFLO では OpenFlow 環境においてレイヤ 7 情報に基づく経路選択を可能にしている．したがって，残された問題は OpenFlow 環境におけるサーバ間での TCP 通信の引継ぎだけである．本節ではこれを可能とする OpenFlow の機能拡張について述べる．

3.1　想定するネットワーク構成と動作

本論文では図 2 に示すように 1 基の OpenFlow スイッチに複数のサーバが接続されているネットワーク構成を想定している．全てのサーバは同一の IP アドレスが割り当てられており，OpenFlow によりクライアントがどのサーバと通信するかが決定される．この構成において OpenFlow スイッチおよびコントローラは以下のように動作する．なお，簡略化のためにクライアントはサーバと TCP 通信を行い，コネクション確立後の最初のクライアントからのパケットに基づいてサーバの選択を行うものとする．

(1) クライアントはサーバに対して SYN パケットを送信する．OpenFlow スイッチはこれを Packet In メッセージにより OpenFlow コントローラに送る．
(2) OpenFlow コントローラは SYN パケットを記録し，フローエントリを登録せずに 1 つのサーバ（この例では Server1 とする）に中継させるように OpenFlow スイッチに Packet Out メッセージを送出する．
(3) Server1 はクライアントとの間で 3 ウェイハンドシェークを継続する．これらのパケットも SYN パケットと同様に OpenFlow コントローラで記録され，OpenFlow スイッチで中継される．
(4) クライアントは Server1 との間でコネクションを確立した後，ペイロードを含む最初のパケットを Server1 に送信する．OpenFlow コントローラはこのパケットを Packet In メッセージにより受け取ると，適切なサーバを選択する．もしそのサーバがクライアントとの間で 3 ウェイハンドシェークを確立したもの（Server1）であれば，(5) に進む．そうでなければ (6) に進む．
(5) OpenFlow コントローラは OpenFlow スイッチに当該フローのパケットがクライアントと Server1 との間で通信できるようにフローエントリを作成し，ペイロードを含む最初のパケットを Server1 に送信するように OpenFlow スイッチに Packet Out メッセージを送る．これ以降は OpenFlow コントローラを介在することなくクライアントと Server1 との間で通信を行う．記録していた 3 ウェイハンドシェークのパケットは破棄する．
(6) OpenFlow コントローラは選択したサーバ（Server2）との間で 3 ウェイハンドシェークを再現する．その際，OpenFlow コントローラが作成する 3 ウェイハンドシェークの最後の ACK パケットでは ACK 番号が Server2 から送られる SYN+ACK パケットのシーケンス番号（ISN）に合わせるようにする．その後，OpenFlow コントローラは OpenFlow スイッチに当該フローのパケットがクライアントと Server2 との間で通信できるようにフローエントリを作成し，ペイロードを含む最初のパケットを Server2 に送信するように OpenFlow スイッチに Packet Out メッセージを送る．これ以降は OpenFlow コントローラを介在することなくクライアントと Server2 との間で通信を行う．また，OpenFlow コントローラは Server1 に対して RST パケットを送出し，Server1 はクライアントとの通信を終了する．

なお，上記の動作において，3 ウェイハンドシェークの際に Server1 ではなく OpenFlow コントローラがクライア

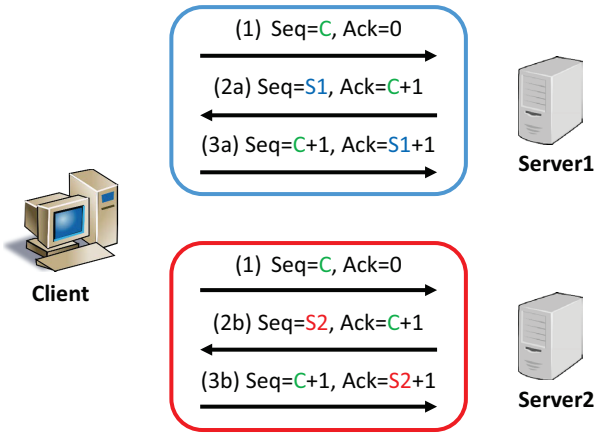

図 3 2 台のサーバを切り替える際のシーケンス番号および ACK 番号の変化

ントとの間で TCP コネクションが確立されるように応答し，サーバを選択した後に上記 (6) の手順を実行する方法も考えられる．この方法では Server1 に対して OpenFlow コントローラが RST パケットを送出する必要がなくなり，また Server1 でもクライアントとの接続記録が残らない利点があるが，一方でどちらのサーバを選択した場合でも 3 ウェイハンドシェークの再現が必要になる点でオーバヘッドが増大することに注意が必要である．

3.2 機能拡張の概要

3.1 節で述べた動作では (6) の段階でクライアントは 3 ウェイハンドシェークの際に得た Server1 の ISN に基づいて送出するパケットの ACK 番号を設定しているため，OpenFlow スイッチがクライアントから受け取ったパケットをそのまま Server2 に中継すると，Server2 は一般的に受け取ったパケットの ACK 番号が自身の期待しているものと大きく異なるため，Server2 はクライアントに RST パケットを送出する．したがって，OpenFlow コントローラは Server1 と Server2 との間で発生する ISN の違いを吸収するように動作する必要がある．2 台のサーバを切り替える際に発生するシーケンス番号，ACK 番号の変化を**図 3** に示す．

もしクライアントとの通信相手となるサーバが選択されるまでにクライアントと Server1 との間でやり取りされるパケットのペイロード長がクライアントと Server1 との間でやり取りされるパケットのものと同じであれば，図 3 における S1 と S2 の差を吸収できるように OpenFlow スイッチが動作すれば，上記 (6) の動作において Server2 はクライアントに RST パケットを送出せず，クライアントと Server2 との間で通信が継続されることになる．そこで，本論文では上記の動作を OpenFlow スイッチが行えるようにシーケンス番号や ACK 番号を OpenFlow スイッチが増減できる機能を持つように拡張することを提案する．すなわち，**図 4** に示すように，OpenFlow スイッチはシーケンス番号あるいは ACK 番号を指定されたオフセットで修正する機能を持つ．具体的には，OpenFlow スイッチには指定されたオフセット Offset $= S2 - S1$ がパラメータとして与えられるフローエントリが登録可能で，Server2 からクライアントへのパケットに対してはシーケンス番号からオフセットを減算した値を用い，逆にクライアントから Server2 へのパケットの ACK 番号にはオフセットを加算した値を用いるようにすれば良い．

3.3 提案機能の適用範囲

提案機能は原則として 3 ウェイハンドシェーク後のクライアントからの通信内容に基づいて実際に通信を行うサーバを選択する．しかし，状況によってはサーバの選択が困難な場合が存在する．本節では提案機能の適用範囲について考察する．

まず，サーバを選択する前に行われる通信は Server1 でも Server2 でも同じ内容である必要がある．したがって，例えば SSL/TLS による暗号化通信において Server1 と Server2 が異なる暗号鍵を用いるようにネゴシエーションを行う場合には提案機能は適用できない．これを実現するには，Server1，Server2，コントローラの 3 者間で共通の秘密鍵を持ち，コントローラはクライアントからの通信内容を復号したうえでサーバを選択する必要がある．

一方，提案機能は実際に通信を行うサーバを決定するまではフローエントリを登録せずに Packet In メッセージを用いてコントローラに通信内容を伝えることが可能である．これにより，たとえば SMTP サーバのように TCP コネクション確立後にまずサーバからクライアントへ通信を行うようなプロトコルであっても，サーバからクライアントへの最初の通信が同じ内容になるように工夫すれば提案機能を適用することが可能である．また，HTTP 通信においてサーバが Cookie を発行する状況であれば，これを確認するまではコントローラはフローエントリを登録せずに当該セッションの Cookie を傍受し，それ以降クライアントからの同一セッションの通信を同一サーバに中継するようにすることも可能である．

さらに，提案機能は基本的には TCP 通信を対象としているが，サーバからクライアントへの通信がサーバに依らず同じであるなどの条件を満たせば UDP 通信に対しても適用可能である．

4. 提案機能の OpenFlow スイッチへの実装

本節では前節で提案した OpenFlow スイッチの拡張機能の実装方法を述べる．この実装では OpenFlow スイッチには Open vSwitch（以下，OVS）[7]2.3.0，OpenFlow コントローラには Ryu[8] を用いた．また，OpenFlow プロトコルのバージョンは最も普及していると思われる 1.3 を用いた．OVS，Ryu はともに OS として Debian をインストー

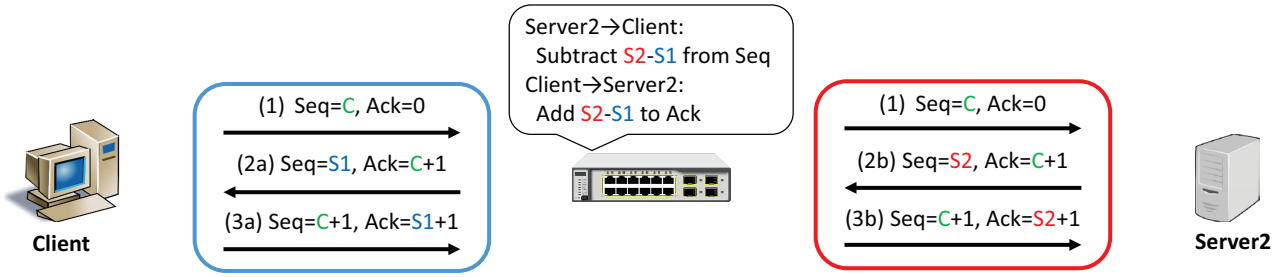

図 4　提案する OpenFlow スイッチの拡張機能

表 1　ヘッダ書換え方法によるスループットの違い

ヘッダ書換え方法	平均スループット	標準偏差
OVS（カーネル空間）	943[Mbps]	0[Mbps]
OVS（ユーザ空間）	9.5	4.5
Ryu（Packet In）	72.1	2.2

ルしたマシン上で実行した．OVS はユーザ空間とカーネル空間の両方で動作させることが可能であるが，高速で動作するカーネルモジュールの改変を容易にするために，OVS が Linux OS と独立している Linux カーネル ver.3.2 を使用し，これに対応する OS として Debian 7 を用いた．

4.1　ヘッダ書換え時のスループット

まず，パケットのヘッダ書換えによるオーバヘッドを評価するため，予備実験として図 2 の構成においてクライアントとサーバ（Server1）との間で通信を行い，IP アドレスならびに MAC アドレスを書き換えた場合のスループットを評価した．通信路の帯域幅は 1Gbps で，スループットの計測には iPerf[9] を用いた [*2]．比較の対象としては OVS をカーネル空間で動作させた場合，OVS をユーザ空間で動作させた場合，全てのパケットを Packet In メッセージで OpenFlow コントローラに送ってそこで書き換える場合の 3 通りとした．その結果を**表 1** に示す．これより，OVS をカーネル空間で実行した場合が最も高いスループットを示し，それ以外の 2 つの場合ではスループットが 10 分の 1 以下に低下していることがわかる．したがって，十分なスループットを得るためには提案機能も OVS のカーネル空間で実装する必要があると言える．

4.2　シーケンス番号・ACK 番号増減機能の実装

OpenFlow コントローラ・スイッチ間でやり取りされるメッセージまでを改変するのは実装上のコストが大きいため，我々は既存のメッセージを利用して新規アクションを呼び出す方法を用いる方法を採用した．ここで用いるのは，TCP ポート番号（以下，ポート番号）を書き換えるアクション（TCP ヘッダ変更アクション）のメッセージである．このアクションがフローエントリとしてテーブルに格

[*2] 特に断りのない限り，デフォルトのパラメータを用いて TCP 通信のスループットを測定した．

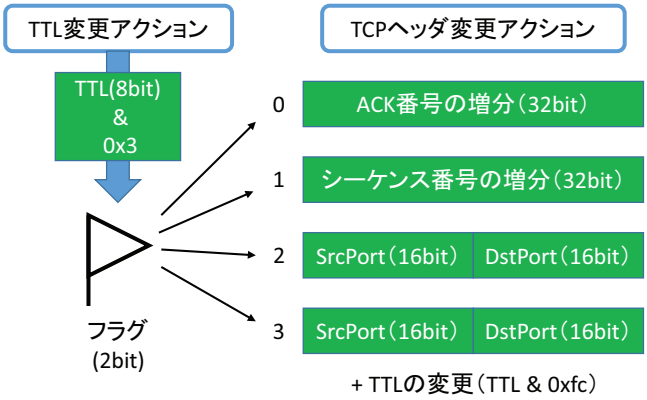

図 5　TTL による TCP ヘッダ変更アクションの選択

納されていたら，ポート番号の代わりにシーケンス番号あるいは ACK 番号を変更するものと解釈する．TCP ヘッダ変更アクションを使用したのは，シーケンス番号や ACK 番号はポート番号と同じく TCP ヘッダ内にあり，さらに宛先ポート番号と送信元ポート番号の合計サイズが 32bit で，これがシーケンス番号・ACK 番号の長さと等しいためである．この性質を使って，シーケンス番号・ACK 番号のオフセット（32bit）を上位下位の 16bit ずつに分割し，それぞれ変更後の宛先/送信元ポート番号を記録すべき 16bit の領域に格納すればメッセージのフォーマットを改変する必要がない．提案するシーケンス番号・ACK 番号増減機能ではオフセットはシーケンス番号と ACK 番号のどちらか一方を増減すればよいため，記録しておくべきオフセットは 1 つで十分である．

しかし，このままではシーケンス番号と ACK 番号のどちらを増減するのか，あるいは本来の宛先/送信元ポート番号の変更を行うのか，区別ができない．そこで，比較的利用頻度が低いと思われる TTL 変更アクションを利用し，変更後の TTL の下位 2bit を動作を指定するフラグとして利用することにした．したがって，シーケンス番号・ACK 番号増減機能を使用する際には必ず TTL 変更アクションを同時に指定することになる．TTL の値に基づく TCP ヘッダ変更アクションの内容を**図 5** に示す．なお，シーケンス番号の操作ではオフセットの減算を行う代わりにオフセットの 2 の補数を増分として登録し，これを加算するようにしている．

表 2　実験環境における各仮想マシンのスペック

種　別	メモリ	OS（linux カーネルバージョン）
OpenFlow Controller	2GB	Debian 8.5 (3.16.0-4)
OpenFlow Switch	2GB	Debian 7.11(3.2.0-4)
Client	2GB	Debian 7.11(3.2.0-4)
Server1	2GB	Linux Mint 18 Sarah(4.4.0-21)
Server2	2GB	Debian 8.9(3.16.0-4)

実際の実装では，TCP ヘッダを操作する関数 set_tcp() においてシーケンス番号を操作する関数 add_tp_seq() と ACK 番号を操作する関数 add_tp_ack() を呼び出すように修正を行った．これらの関数ではポート番号を変更する既存の関数 set_tp_port() と同様にアドレス取得，書き換える値の取得，チェックサム再計算，ヘッダ書き換え，ハッシュ再計算の各処理をこの順で実行するようにした．

なお，この実装方法ではシーケンス番号や ACK 番号の増減に加えてポート番号を書き換えることはできない．これは，もし 2 つのアクションを登録し，一方でシーケンス番号や ACK 番号の増減を行い，もう一方でポート番号の書換えを行うように設定したとしても，OVS の最適化機能により最初に実行されるアクションが冗長と判断されて削除されるためである．これを行うには OVS 内で 2 台のブリッジを定義してそれぞれでシーケンス番号や ACK 番号の増減とポート番号の書換えを行うようにすればよい．

5.　機能拡張した OVS の評価

5.1　評価環境

4.2 節で述べた拡張機能が期待通りに動作するか，またこの実装方法で十分な性能が得られるかどうかを確認するために評価実験を行った．実験環境は図 2 と同様であるが，1 台のコンピュータ（CPU: Intel Core i5-4210M 2.60GHz，メモリ: 8GB，OS:Windows 10 Pro）上で動作する仮想マシン環境ソフト VirtualBox[10] 上に構築した．実験環境における各仮想マシンのスペックを**表 2** に示す．各回線および NIC（Network Interface Card）の帯域幅は 1Gbps に対応するようにした．

5.2　ヘッダ書換え機能の検証

新たに実装したシーケンス番号・ACK 番号の書換え機能が正常に動作するかどうかを検証する実験を行った．この実験では (i) シーケンス番号・ACK 番号を 0x01020304 だけ加減算するように書き換えた場合と，(ii)ACK 番号に加えて下位レイヤ情報である IP アドレスも併せて書き換えた場合に，正しく動作することを確認するようにした．

まず，(i) の結果として，クライアントが送出した SYN パケットを**図 6** に，サーバが受信した SYN パケットを**図 7** に示す．この実験では拡張機能を有効にした状態で TCP 通信を行えるように，図 4 とは異なり，クライアントが送出したパケットに対してシーケンス番号を変更し，サーバが送出したパケットに対してシーケンス番号を変更するように OVS を設定している．図において四角で囲っている部分はこれらの図の間で異なっている箇所を示す．これらの図より，異なっている部分はシーケンス番号とチェックサムのみであり，他の部分は変更されていないことが確認できる．また，同様に SYN パケットへの応答としてサーバが送出した SYN+ACK パケットを**図 8** に，クライアントが受信した SYN+ACK パケットを**図 9** に示す．これらの図よりでは ACK 番号とチェックサムが変更されており，それ以外の部分は変更されていないことが確認できる．

```
4500 003c d481 4000 4006 e2e6 c0a8 0101
c0a8 0102 82d2 abcd 5ec7 62a5 0000 0000
a002 3908 8382 0000 0204 05b4 0402 080a
0007 b624 0000 0000 0103 0305
```

図 6　クライアントが送出した SYN パケット

```
4500 003c d481 4000 4006 e2e6 c0a8 0101
c0a8 0102 82d2 abcd 5fc9 65a9 0000 0000
a002 3908 e167 0000 0204 05b4 0402 080a
0007 b624 0000 0000 0103 0305
```

図 7　サーバが受信した SYN パケット

```
4500 003c d481 4000 4006 e2e6 c0a8 0101
c0a8 0102 82d2 abcd 5fc9 65a9 0000 0000
a002 3908 e167 0000 0204 05b4 0402 080a
0007 b624 0000 0000 0103 0305
```

図 8　サーバが送出した SYN+ACK パケット

```
4500 003c d481 4000 4006 e2e6 c0a8 0101
c0a8 0102 82d2 abcd 5ec7 62a5 0000 0000
a002 3908 8382 0000 0204 05b4 0402 080a
0007 b624 0000 0000 0103 0305
```

図 9　クライアントが受信した SYN+ACK パケット

次に，(ii) の結果として，クライアントが送出した通信中の TCP パケットを**図 10** に，これをサーバが受信した際のパケットを**図 11** に示す．この実験では図 4 と同様にクライアントが送出したパケットに対して ACK 番号を変更するように OVS を設定している．これらの図より，今回は ACK 番号に加えて宛先 IP アドレスも変更されているが，その他の部分は変更されていないことが確認できる．

以上の結果から，拡張機能であるシーケンス番号・ACK 番号の書換え機能は設計通りに動作し，また，下位レイヤ情報である IP アドレスの書換えと両立できることが確認された．

5.3　OVS のスループットの評価

次に 4.2 節で述べた拡張機能の実装方法で十分な性能が得られるかどうかを検証するため，拡張機能を組み込んだ

```
4500 0098 a16f 4000 4006 159d c0a8 0101
c0a8 0102 82e0 abcd 2b75 f8e3 2293 cf64
8018 01c9 83de 0000 0101 080a 000f 84a1
000f 5910 3132 3334 3536 3738 3930 3132
```

図 10 クライアントが送出した通信中の TCP パケット

```
4500 0098 a16f 4000 4006 159d c0a8 0101
c0a8 0103 82e0 abcd 2b75 f8e3 b740 1e0d
8018 01c9 86dc 0000 0101 080a 000f 84a1
000f 5910 3132 3334 3536 3738 3930 3132
```

図 11 サーバが受信した通信中の TCP パケット

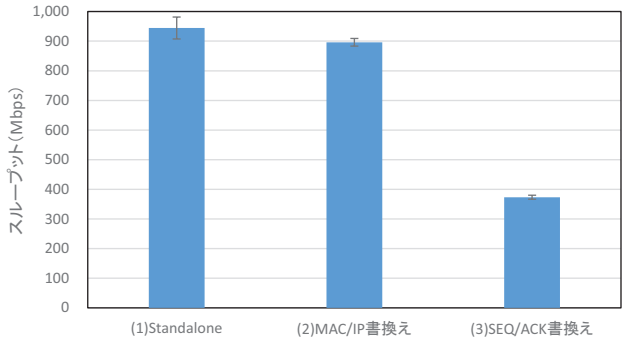

図 12 OVS の様々な動作に対するスループットの違い

OVS のスループットを測定した．本実験では，比較対象を含む以下の 3 つの場合のスループットを iPerf を用いて 10 秒間計測し，これを 5 回繰り返して平均スループットおよび標準偏差を求めた．

(1) Ryu を起動せずに，OVS の設定を Standalone として通常のレイヤ 2 スイッチ同様に動作させる．
(2) OVS で IP アドレスと MAC アドレスを書き換える．
(3) OVS でシーケンス番号と ACK 番号を書き換える．

その結果を図 **12** に示す．この図からわかるように，Standalone の動作では回線の帯域幅 1Gbps に近いスループットが得られている．また，OVS で IP アドレスと MAC アドレスを書き換える動作を行った場合でも Standalone の場合と比べて 5%程度の低下が見られるものの，実用上十分なスループットが得られている．一方，OVS でシーケンス番号と ACK 番号を書き換える場合では Standalone の場合と比べて 4 割程度のスループットしか得られていない．シーケンス番号や ACK 番号の書換え処理は IP アドレスと MAC アドレスの書換え処理と本質的な部分で違いはないため，予想としては 5%程度の低下を見込んでいたが，予想とは大きく異なる結果となった．この原因としては，本実装で既存のメッセージの処理にシーケンス番号や ACK 番号の書換え処理を追加したことで TTL の変更が必須となり，実質的に各パケットの処理内容が 1 つではなく 2 つになっただけでなく，TCP ヘッダを操作する関数 set_tcp() の処理内容も複雑になったため，その分オーバヘッドが大きくなったことが挙げられる．

6. まとめ

本論文ではコマンドの引数などレイヤ 7 の情報に応じたサーバ選択を可能にするため，OpenFlow 環境において TCP コネクションの確立後にホスト変更機能ができるような OpenFlow スイッチの機能拡張を提案した．この機能拡張では切替え前後のホスト間の ISN の違いを吸収できるようにするために，TCP のシーケンス番号ならびに ACK 番号を両ホストの ISN の差に相当するオフセット値だけ増減することができる．

しかし，シーケンス番号・ACK 番号を変更する場合の OVS のスループットを計測したところ，MAC/IP アドレスを書き換える場合と比較して約 4 割のスループットしか得られず，改善の余地があることが分かった．その原因としては既存のメッセージ処理の流用が考えるため，今後の課題としては専用のメッセージを用意し，既存のメッセージ処理と分離することが挙げられる．

参考文献

[1] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker and Jonathan Turner: “OpenFlow: Enabling Innovation in Campus Networks”, *ACM SIGCOMM Computer Communication Review*, Vol. 38, No. 2, pp.69–74, April 2008.

[2] Open Network Foundation: OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06), available from ⟨https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf⟩ (accessed 2017-09-12), March 2015.

[3] Richard Wang, Dana Butnariu and Jennifer Rexford: “Openflow-Based Server Load Balancing Gone Wild”, *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE ’11)*, USENIX Association, March 2011.

[4] Muhammad Nugraha, Isyana Paramita, Ardiansyah Musa, Deokjai Choi, and Buseung Cho: “Utilizing OpenFlow and sFlow to Detect and Mitigate SYN Flooding Attack”, *Journal of Korea Multimedia Society*, Vol. 17, No. 8, pp.988–994, August 2014.

[5] Haopei Wang, Lei Xu, and Guofei Gu: “FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks”, *DSN ’15 Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp.239–250, June 2015.

[6] Vasaka Visoottiviseth, Suthasinee Lertviriyasawat, Peerada Suppiyatrakoon, Pattarajit Chitkornkitsil, and Nariyoshi Yamai: “REFLO: Reactive Firewall System with OpenFlow and Flow Monitoring System”, *Proceedings of The 2017 IEEE Region 10 Conference (TENCON 2017)*, #1570376415, pp.2273–2278, November 5-8, 2017.

[7] A Linux Foundation Collaborative Project: Open vSwitch (online), available from ⟨http://www.openvswitch.org/⟩ (accessed 2018-09-12), 2016.

[8] Ryu SDN Framework Community: Ryu SDN Framework (online), available from ⟨https://osrg.github.io/ryu/⟩, (accessed 2018-09-16), 2017.
[9] —: iPerf - The TCP, UDP and SCTP network bandwidth measurement tool (online), available from ⟨https://iperf.fr/⟩ (accessed 2018-09-16), 2017.
[10] Oracle: Oracle VM VirtualBox (online), available from ⟨https://www.virtualbox.org/⟩ (accessed 2018-09-16).