マルチテナント型SDN仮想化基盤の設計と実装

Higuchi, Shun / 樋口, 俊

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
13
(開始ページ / Start Page)
1
(終了ページ / End Page)
6
(発行年 / Year)
2017-03-31
(URL)
https://doi.org/10.15002/00021529

マルチテナント型 SDN 仮想化基盤の設計と実装
Design and Implementation of Multi-Tenant SDN Hypervisor

樋口　俊
Shun Higuchi
法政大学大学院　情報科学研究科　情報科学専攻
E-mail: shun.higuchi.6j@stu.hosei.ac.jp

Abstract

Cloud services that virtualize existing IT infrastructures in data centers are widely used by governments, universities, and companies. Multi-tenancy is an indispensable feature for data centers to provide a large number of isolated networks to different organizations. OpenFlow is a core technology of software defined networking (SDN) and is useful for centrally managing and controlling these networks; however, SDN is used only at the management level. It is desirable to make the flexible features of SDN/OpenFlow available to users' virtual networks. FlowVisor provides virtualized multi-tenant OpenFlow networks by coordinating multiple controllers, but it is unable to prevent conflicts among the control rules of individual virtual networks. Administrators of each tenant thus need to design the specifications of each virtual network carefully. In this paper, we propose a verification based virtualization system for multiple tenants' Openflow networks. It enables users of virtual network, that are administrators of the tenant's network, to design their own virtual networks without considering conflicts with other tenants. Our system provides a practical scheme to abstract and create virtual network topology which is the target of control for the tenant's Openflow controller.

1　はじめに

サーバ仮想化技術の発展により，組織が必要とする IT インフラストラクチャをデータセンタ上で仮想化しインターネット経由で提供する IaaS(Infrastructure as a Service) などのクラウドコンピューティングサービスが普及している．このようなサービスを提供するデータセンタでは，物理リソースを複数企業に共有させるマルチテナントへの対応が必要となる．その中でも，マルチテナントネットワークでは 1 つの物理ネットワークを複数のテナント用仮想ネットワークへと論理的に分割し，それぞれの仮想ネットワーク内で行われる通信を分離する必要がある．これらを実現するネットワーク仮想化技術としては，従来は VLAN 技術の利用が一般的であった．IaaS 提供者は各テナントネットワークに VLAN-ID を割り当てることで 1 つの物理ネットワークを複数のレイヤ 2 ネットワークに分割し，各テナントのネットワーク管理者は割り当てられた複数のレイヤ 2 ネットワークを組み合わせて自由にレイヤ 3 ネットワークを構築していた．VLAN を利用した手法ではネットワーク構成の変更の度に IaaS 提供者が必要な全ネットワーク機器に対して VLAN の設定を変更することが必要になる．従来に比べ動的に仮想ネットワークや仮想マシンの動的な増減が頻発するクラウド環境では，構成の変更に対応したより柔軟な仮想ネットワークの構築・管理手法が必要とされる．

この要求を満たす技術として，近年注目されている Software-Defined Networking (SDN)[1] の代表的アーキテクチャである OpenFlow[2] が挙げられる．OpenFlow では経路制御を行うコントローラとデータ転送を行うスイッチを分離することで，柔軟な経路制御とネットワークの集中管理が可能となっている．OpenFlow ではコントローラからスイッチに書き込むフローエントリにおいて VLAN-ID の認識・書き換えなどが指定できるため，IaaS 提供者は構成変更に伴う VLAN 管理を 1 つのコントローラで集約的に管理することができる．一方で，SDN 技術の一つの特徴である「ソフトウェアによる柔軟なネットワーク制御機能」をテナント側が使おうとすると，管理レベルの制御と競合してしまう．このように SDN で管理された IaaS 基盤上で各テナントによる柔軟な制御を可能とするためには，各テナントが発する処理の要求を競合を回避しながら処理する仕組みが必要となる．

複数の OpenFlow コントローラの要求を処理する技術としては FlowVisor[3] が挙げられる．FlowVisor はコントローラとスイッチ間にプロキシとして配置され，それらの間で制御メッセージの交換を管理する．これによって 1 つの OpenFlow ネットワークを複数のコントローラで個別に制御することが可能となっている．FlowVisor では予めネットワーク空間をフロースペースという単位に分割し，各テナントユーザが自身のコントローラに割り当てられたフロースペースに書き込むフローエントリを設定する．これによって複数の仮想 OpenFlow ネットワークを異なるテナントコントローラで制御することが出来ている．しかし，FlowVisor では各フロースペース間の競合検証を行わずに衝突の回避を IaaS 管理者によるネットワーク設計に任せているため，テナントユーザにとっては自由な設計が許されていなかった．

本研究では IaaS 環境においてテナント毎に自由なネットワークの設計を可能にする SDN 仮想化基盤の実現を目的とする．これを実現するために，テナントに対して提供するネットワークトポロジの抽象化，テナントが自由に設計したネットワーク定義の衝突検証と管理，物理トポロジに対するテナントネットワークの柔軟なマッピング手法といった機能が必要になる．提案する機能によって，テナントは自由にネットワークを設計可能であると共に，SDN による任意の制御を行った場合にもテナントネットワーク間の分離性が保証されていることを示す．

Supervisor: Prof. Toshio Hirotsu

2 SDN/OpenFlow

クラウド環境を中心とする次世代基盤技術として注目を集めているのが，Software-Defined Network の代表的アーキテクチャの 1 つとして標準化が進む OpenFlow である．OpenFlow はコントローラプレーンとデータプレーンが分離され，ネットワークの経路制御を担う OpenFlow コントローラによって，パケットの転送を行う OpenFlow スイッチを一元管理する中央制御型アーキテクチャとなっている．

ソフトウェアであるコントローラにおいてレイヤ 1〜4 のマッチ条件とパケットへの処理の組をフローエントリとして定義し，これに従ってスイッチがパケットを処理することで柔軟な経路制御が可能である．コントローラとスイッチ間は，データネットワークとは別に OpenFlow チャンネルと呼ばれる TCP/IP を用いた制御ネットワークによって接続され，OpenFlow メッセージと呼ばれる制御情報のやりとりが行われる．コントローラはこの OpenFlow メッセージを通して，フローエントリの書込みなどのスイッチ制御を行う．OpenFlow ではソースルーティングやマルチパス転送といった柔軟な経路制御に加えて，ネットワークの仮想化では VLAN-ID の管理性向上だけでなく，マッチ条件として指定したヘッダ情報を用いてネットワークの論理的な分割が可能である．

これらの利点が存在する一方で従来の OpenFlow 技術では，1 つの OpenFlow ネットワークを複数のコントローラで個別に制御する，1 つの OpenFlow ネットワークを複数の仮想 OpenFlow ネットワークに分割する，などの OpenFlow ネットワークそのものを仮想化する仕組みが実装されていないという課題があった．この問題から，IaaS を提供しているマルチテナント型データセンタなどにおいて各テナントがそれぞれのコントローラと OpenFlow ネットワークを制御するといった利用形態が不可能だった．

3 FlowVisor

FlowVisor は，コントローラとスイッチ間を接続する OpenFlow チャンネル上に配置され，コントローラからスイッチを制御するのに必要な OpenFlow メッセージを転送する透過型プロキシとして動作する．まず，FlowVisor の管理者が予めそれぞれのテナントで利用可能なネットワーク空間をフロースペースとして定義しておく必要がある．フロースペースでは，テナントネットワークの名前を示すスライス名と OpenFlow スイッチの DPID，そして MAC アドレスや IP アドレス，トランスポート番号など OpenFlow のフローエントリ中で利用可能なレイヤ 1 からレイヤ 4 までのマッチフィールドと優先度の空間を定義する．また，それぞれのフロースペースで定義されているネットワークの空間は重複せず独立していることを前提としており，管理者が注意深くフロースペースの定義を個々のスイッチに対して行う必要がある．

FlowVisor の利用形態としては，まず管理者が各テナントが利用できるネットワーク空間をフロースペースとして定義した後に，その情報を何らかの形でそれぞれのテナントユーザに提示する．各テナントユーザは，FlowVisor の管理者から提示された仮想 OpenFlow ネットワークのトポロジ情報とフロースペース情報に基づいて，フローエントリとそれを書き込むコントローラを作成し FlowVisor に接続することでテナントネットワークが制御可能となる．

FlowVisor では，各テナントに自由にネットワークを設計させることを想定してそれぞれが定義したフロースペースをそのまま利用すると，他のフロースペースと衝突するようなフローエントリが書き込まれて意図しないトラフィック制御が行われる問題が存在していた．また，この問題を発生させないために FlowVisor 管理者にネットワーク設計を委ねるという方針は，テナントユーザにとっての自由なネットワーク設計を制限しており，IaaS 環境に求められている自由かつ柔軟なマルチテナントネットワークの提供とは異なるものだった．

4 マルチテナント型 SDN 仮想化基盤

本研究では，テナントごとに自由なネットワーク設計と制御を可能にする OpenFlow ネットワークの仮想化技術を提案する．図 1 に示すように OpenFlow ハイパーバイザ上に 3 つの機構を提供する．まずテナントに対して十分な経路の冗長性を確保した OpenFlow ネットワークの抽象化機構を提案する．この機構によって，テナントが利用しない部分の物理トポロジを隠蔽した上で，そのテナントに必要とされる部分を抽象化した仮想トポロジがテナントのネットワーク設計者に対して提供される．次に，自由な設計と自動検証を支援する新たなフロースペースの概念を導入し，このフロースペースのアドレス空間に対して重複部分を検証・管理する機構を提案する．各テナントネットワークで利用するアドレス空間の組み合わせそのものをフロースペースとして定義し重複を検証・管理することで，テナント間でフローエントリが衝突しうる部分について予め管理しておくことができる．加えて，フローエントリの衝突が起きうるテナントネットワーク間を分離するために，Virtual eXtensible Local Area Network (VXLAN)[4] を用いた仮想トポロジの構築機構を提案する．この機構では，物理ネットワークと仮想テナントネットワークの柔軟なマッピングを行うだけでなく，フローエントリの衝突を効率的に回避することができる．これらの機構により本研究では，各テナントに対してテナントネットワークの自由な設計を許すとともに，任意の制御に対してテナントネットワーク間の通信の分離を実現している．

4.1 テナントネットワークの抽象化

テナントが新たに追加された際にはそのネットワーク設計を支援するために，テナントに利用可能なネットワークトポロジが提示される．この時，テナントに対して物理トポロジ全体を表示するのではなく，そのテナント用に抽象化された仮想トポロジが提示される．ここでは，ホストが接続されているエッジスイッチと外部に通信するゲートウェイスイッチ間の接続関係をベースとして，物理トポロジから複数の経路を抜き出すことで抽象化し仮想トポロジを構成する手法について提案する．この抽象化手法では，テナントが負荷分散や冗長化に利用する冗長経路を常に確保するために，サイクル構造となる仮想トポロジを構成する．

仮想トポロジの構成手順は以下の通りである．ここではスイッチ間のリンクを枝 $E = \{e_1, e_2, ...e_n\}$ として，トポロジのグラフを枝集合 $G = (E)$ で示す．

1. グラフ G においてゲートウェイスイッチから各エッジス

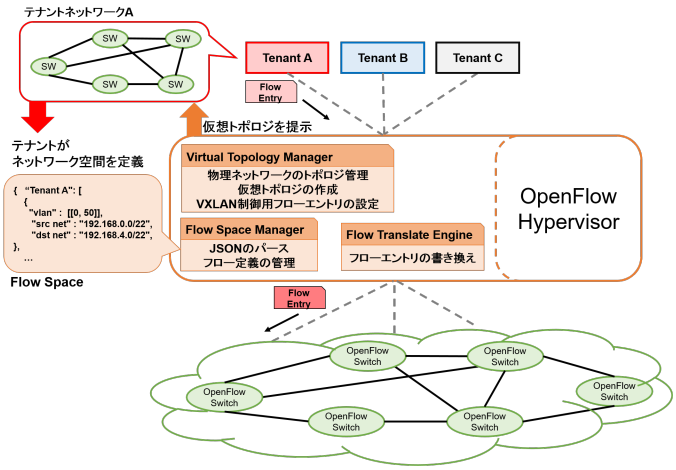

図 1　マルチテナント型 SDN 仮想化基盤

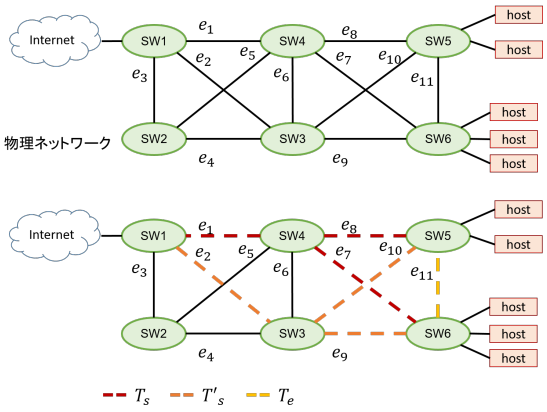

図 2　物理トポロジの抽象化

イッチに対して幅優先探索による最短経路を探索し，木 T_s を構成する．例として図 2 ではゲートウェイ SW1 とエッジスイッチ SW5・SW6 間の経路として $T_s = \{e_1, e_7, e_8\}$ が作成される．

2. 木 T_s とグラフ G の排他的論理和をとることで T_s を除外したグラフ G' を構成する．このグラフ G' においてもゲートウェイスイッチから各エッジスイッチ間の最短経路を探索し木 T'_s とする．この時，経路が存在しない場合にはそのエッジスイッチに対する探索を打ち切る．例として図 2 では，T_s を用いて $G' = \{e_2, e_3, e_4, e_5, e_6, e_9, e_{10}, e_{11}\}$ が構成され，探索すると $T'_s = \{e_2, e_9, e_{10}\}$ となる．
3. グラフ G においてエッジスイッチ同士を接続する最短経路群を幅優先探索で求めて枝集合 T_e とする．例として図 2 では，$G = \{e_1, e_2, ...e_{11}\}$ を用いて $T_e = \{e_{11}\}$ となる．
4. T と T，e の論理和をとって仮想トポロジのグラフ G とする．例として図 2 では，$T_s = \{e_1, e_7, e_8\}$，$T'_s = \{e_2, e_9, e_{10}\}$，$T_e = \{e_{11}\}$ の論理和から $G = \{e_1, e_2, e_7, e_8, e_9, e_{10}, e_{11}\}$ となる．

ゲートウェイスイッチとエッジスイッチ間において最短経路とその冗長経路を確保した上で，エッジスイッチ同士は相互に接続することで冗長性に優れるリングと部分的なメッシュトポロジが構成される．また，ゲートウェイスイッチが 2 つ以上存在する場合にはそれぞれに対して構成手順を適用して木を作成し，最後に論理和をとって合成した木を仮想トポロジとする．

4.2　フロースペースの定義

FlowVisor のフロースペースでは各テナントが制御可能なスイッチ毎に定義を行っていたが，この手法のフロースペースでは 1 つのテナントネットワーク全体に対して利用可能なアドレス空間の組み合わせを定義する．1 つのフロースペースは複数のルールによって構成され，1 つのルールは表 1 に示されるようにルール ID，フロースペース名，テナントが利用可能なマッチングフィールドの並びから成っている．マッチングフィールドにはネットワークを定義する実運用上で必要な，VLAN ID，Src/Dst IP アドレスという L2・L3 の 3 種類のヘッダ情報を指定することができる．これらの内，Src/Dst IP アドレスについては管理性の面から最小を 27bit とするサブネット空間として設計を行う．1 つのフロースペースは，フロースペース名とそれぞれのフロー定義の集合で記述される．フロー定義はマッチフィールドの各要素毎に記述し，1 つのフロー定義は各フィールドの要素の AND として定義される．1 つのフロースペースは 1 つ以上のフロー定義で表すこととして，複数のフロー定義はいずれかに合致するフローエントリが許可され OR として機能する．これによって，各テナントではこのフロースペースで指定されたアドレス空間の組み合わせを利用することが出来る．定義例をまとめた表 1 において，定義例 2 では以下のようなアドレス空間を形成している．

- VLAN ID = 100, Src IP = 192.168.64.0/20, Dst IP = 192.168.64.0/20
- VLAN ID = 101, Src IP = 192.168.64.0/20, Dst IP = 192.168.64.0/20

このフロースペースを割り当てられたテナントは，これら 2 種類の組み合わせをフローエントリのマッチフィールドに用いてネットワークを制御することが出来る．

4.3　フロースペースの重複検証

各テナントのフロースペースを分解して作成されたフロー定義群について重複の検証と管理を行う．この重複検証では，単純に 2 つのフロー定義間でアドレス空間の包含関係について検証を行い，完全な包含関係にあるフロー定義をフローエントリの衝突が起こり得るフロー定義として重複関係を管理する．重複の例として表 1 のフロースペースについて定義例 1 と定義例 2 を分解したフロー定義群の中には，以下の 2 つのフロー定義が存在している．

- 定義例 1：VLAN ID = 100, Src IP = 192.168.64.0/22, Dst IP = 192.168.64.0/22
- 定義例 2：VLAN ID = 100, Src IP = 192.168.64.0/20, Dst IP = 192.168.64.0/20

この 2 つのフロー定義は完全な包含関係にあるため，重複があるフロー定義として管理される．このような重複しているフ

表 1　フロースペースの定義例

Rule ID	Space Name	VLAN	Src IP	Dst IP
1	定義例 1	0〜100	192.168.64.0/22	192.168.64.0/22
2	定義例 2	100, 101	192.168.64.0/20	192.168.64.0/20

表 2　特殊な論理ポート

Entry	Table ID
IN_PORT	パケットの入力ポート
ALL	入力以外の全てのポート
CONTROLLER	コントローラに明示的に送信

ロー定義においてテナントネットワークのトポロジに重なりがある場合，テナント間でフローエントリの衝突が発生する可能性があるためトポロジの分離が必要となる．

5　VXLAN を用いた仮想トポロジの構築

OpenFlow では，標準の機能として VXLAN の ID である VNI をマッチフィールドで指定できる一方で，パケットのカプセリング/復号化自体は行うことはできない．これに対して，OpenFlow の拡張仕様である Nicira Extentions ではパケットに対する VNI の付与など VXLAN によるカプセリングと復号化をフローエントリを用いて行うことが出来る．本研究では，この機能を利用し VXLAN を用いた仮想トポロジの構築手法を提案する．OpenFlow の各バージョン間では互換性が無いこととは異なり，この Nicira Extentions によるフローエントリと通常のフローエントリは同時に利用可能なため，テナントコントローラでは標準仕様のフローエントリを書き込ませながら OpenFlow ハイパーバイザからは Nicira Extentions 仕様のフローエントリによって制御を行うことが可能になる．VNI には 20bit の値を表現できるため，この値をフロースペースが競合しているテナントに対して一意に割り振ることで約 1600 万以上のテナントを収容することができる．

ここでは，VNI と管理用のフローエントリを用いた 2 段階の制御によって仮想トポロジを構築する．まず，テナントのエッジスイッチそれぞれに対して，表 3 のようなホストからのパケットをカプセリングするフローエントリと，ホストに向けて送出されるパケットを復号化するフローエントリの 2 種類を設定する．カプセリング用のフローエントリでは，パケットとのマッチ条件にホストが接続している物理ポートを指定し，2 段階のアクションによってカプセリングを行う．2 段階のアクションでは，まず Set-Tunnel アクションによってパケットにテナントの VNI を付与し，続いて Move アクションによってカプセル化ヘッダとして元のヘッダ情報をコピーして付与する．カプセル化ヘッダに元のヘッダと同じ情報を利用することで，後述のフローエントリ変換の際に VNI のみの書き換えで済む．復号化用のフローエントリでは，パケットとのマッチ条件に VNI を指定してアクションで VNI を 0 に指定することで復号化を行う．これらについて，カプセリングを行うフローエントリは最も早いフローテーブルでマッチし，復号化は最後のフローテーブルでマッチするというように指定して書き込む．最後に，テナントから書き込まれるフローエントリに対してマッチフィールドに VNI を追加するように書き換えを行う．

5.1　インストラクション/アクションの検証と書き換え

OpenFlow 1.3 ではパケットに対する処理として，指定した物理ポートからの送出，各ヘッダ情報の書き換えなどの 17 種類のアクションが定義されている．フローエントリ中では，これらから実行したいアクションの集合と複数のフローテーブル

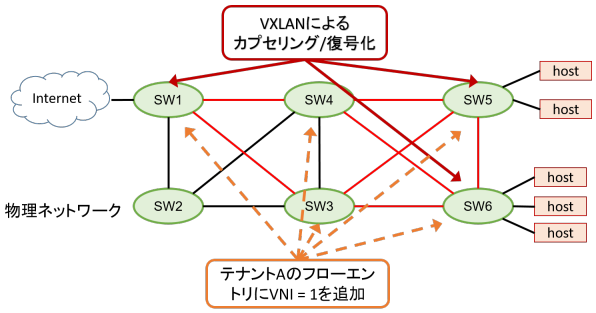

図 3　VXLAN による仮想トポロジの構築

間の遷移処理をインストラクションとして記述することで柔軟な通信制御を実現している．トラフィックの分離性を保証するために，各テナントが書き込むフローエントリに対してアクションで利用している物理ポート番号やヘッダ情報についても検証を行い，フロースペース上の記述や仮想トポロジの情報に違反していた場合には書き換えを行う．

また，スイッチに書き込まれるフローエントリは各テナントのコントローラからのものだけでなく，提案する OpenFlow ハイパーバイザによるものも存在している．OpenFlow ハイパーバイザによるフローエントリは，5 節のように衝突回避を行うためにテナントのフローエントリより前にマッチする，全ての処理の最後にマッチするなどの順番を考慮する必要がある．このため，必ずマッチさせる必要がある管理用フローエントリのためにいくつかのフローテーブルを管理用として確保する．その上で，テナントがフローエントリを書き込むフローテーブルについても検証を行うことで，管理用フローエントリに影響を及ぼすような記述を制限する．

5.1.1　アクションの検証と書き換え

OpenFlow 1.3 のアクションの中でも以下の 2 種類については，アクション中で指定されているパラメータを検証してフロースペースや仮想トポロジとして設計されている値を外れないように制限や書き換えを行う必要がある．

- OFPAT_OUTPUT：指定したポートからパケットを送出
- OFPAT_SET_FIELD：指定したヘッダ情報を書き換え

この中で OFPAT_SET_FIELD アクションについては，設計されたフロースペースのアドレス空間と仮想トポロジ用に割り当てられた VXLAN ID 以外を指定したフローエントリをテナントが書き込もうとした場合に，OpenFlow メッセージごと破棄してテナントのコントローラに通知を行う．

この一方で，OFPAT_OUTPUT アクションではスイッチの物理ポート番号以外にも表 2 に示した特殊な論理ポートがパラメータとして指定できるため，これらについて制限と書き換えを行う必要がある．表の論理ポートから ALL を指定した場合

表 3　VXLAN を用いた仮想トポロジ構築フローエントリ

Entry	Table ID	Match Field	Action
カプセル化	0	in_port = ホストの接続ポート	Set-Tunnel: テナントの VNI Move: カプセル化前と同じヘッダ
復号化	255	tunnel_id_nxm = テナントの VNI	Set-Tunnel: 0

には入力以外の全ポートからフラッディングが行われるが，テナントがこのポートをアクションで指定した場合には，そのテナントが制御可能なポート番号の組み合わせに応じて 1 つのフローエントリを複数のフローエントリに展開してから書き込みを行う必要がある．例として，ある 8 ポートの物理スイッチ上でテナント A は 1,2,3 の物理ポートを利用可能な場合には，以下の 3 通りの入力/出力ポートの組み合わせが存在する．

- IN_Port = 1　Output: 2, 3
- IN_Port = 2　Output: 1, 3
- IN_Port = 3　Output: 1, 2

このため，元のアクションに ALL を指定したフローエントリを破棄する代わりに，上記の 3 パターン分のフローエントリに展開して書き込みを行う．また，仮想トポロジ上で制御可能な物理ポート以外が指定された場合には，再びフローエントリを書き込む OpenFlow メッセージごと破棄してテナントのコントローラに通知を行う．

5.1.2　管理用フローテーブル

OpenFlow 1.3 におけるフローテーブルの ID は 0〜255 となっており，5 節で述べているような必ず最初にマッチするフローエントリや最後にマッチするフローエントリをシステム側で書き込む必要がある場合には，Table ID=0 と 255 のフローテーブルを管理用に予約しておく必要がある．このため，テナントが実際に利用可能な Table ID の値は 1〜254 に制限する．テナントが書き込んだフローエントリが Table ID=0 もしくは 255 の場合には，そのフローエントリを破棄する．

6　実装と評価

6.1　フロースペースマネージャ

フロースペースマネージャでは与えられたフロースペースの定義を保持し，フローエントリが衝突する可能性のあるフロースペースを予め調査しておく．ここではフロースペースの定義を入力とし，これを解析してフロースペース毎にフローの定義群を保持する．この際に各々のフロー定義について，他のフロースペースのフロー定義と全てのフィールドで重複しているものが衝突が発生する可能性があるフロー定義となる．

フロー定義は，発信元 IP アドレス空間について 24bit プレフィックスのネットワークアドレスをキーとしたハッシュで管理する．この際，フロー定義の発信元 IP アドレス空間が/24 より狭い場合は，それが含まれる/24 ネットワークのネットワークアドレスがキーとなり，/24 より大きいときにはそれが含む全ての/24 ネットワークのネットワークアドレスについて複数のエントリを登録する．

このマネージャを Python 3.6 で実装し，性能評価としてフロー登録のオーバヘッドを測定した．ここでは，衝突を全く含まない 5000 個のフロースペースと，全てについて衝突が発生する 5000 個のフロースペースという二つの場合について 1 個目から 5000 個目までにかかる時間の推移を計測した．Intel Core-i7 3.6GHz, 16GB メモリの計算機で測定した結果を図 4 に結果を示す．衝突が発生するフロースペースの方が遅いが，1 エントリ当たり 0.2ms 程度で処理されている．フロースペースの登録は比較的頻度が低いことを考えれば，十分に実用的な性能が得られることが期待できる．

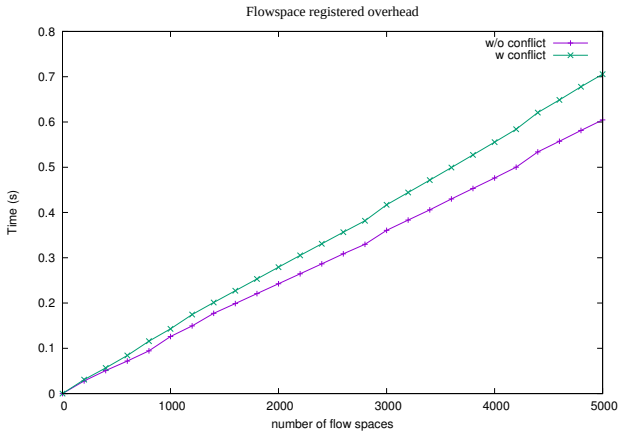

図 4　フロースペース登録の処理時間

6.2　フロー変換エンジン

フロー変換エンジンでは，各テナントコントローラから OpenFlow スイッチに対して書き込むフローエントリを受信し，必要に応じてフローエントリの書き換えを行う．OpenFlow では，コントローラが Flow-Mod メッセージをスイッチに送信することでフローエントリが設定される．フロー変換エンジンはテナントコントローラが送信した Flow-Mod メッセージのパケットをパースし，パケットデータ内のフローエントリに関する情報を書き換えて対象のスイッチに送信する．

この変換エンジンを Python 3.6 と OpenFlow コントローラの開発フレームワークである Ryu SDN Framework 4.16 を用いて実装し，フローエントリの書き換えにかかる処理時間について測定した．この実験では，30,000 個の Flow-Mod メッセージを入力に 2 種類の書き換えパターンについて処理時間の推移を計測している．一つ目のパターンでは，フローエントリ中の OFPAT_OUTPUT アクションで指定した 1 つの物理ポート番号を他の物理ポート番号に書き換えている．二つ目のパターンでは，OFPAT_OUTPUT アクションで指定した論理ポートを 4 つのポート番号に展開した上で書き換えている．この実験を 6.1 節と同じ計算機で行った結果が図 5 である．実装したフロー変換エンジンでは，両方の書き換えパターンにおいて 30,000 個の Flow-Mod メッセージを 13 秒程度で書き換えることができている．これは 1 つの Flow-Mod メッセージを

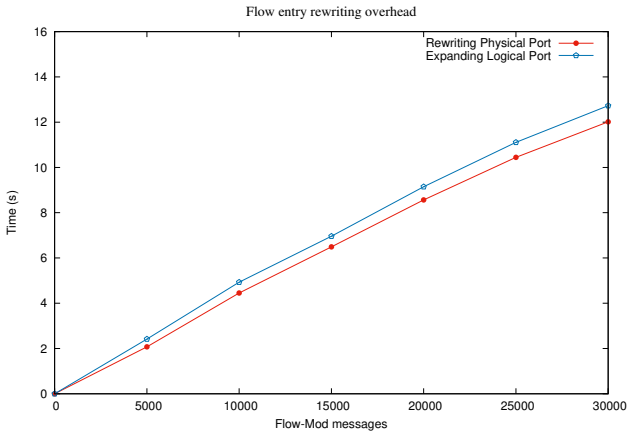

図 5　フローエントリ書き換えの処理時間

平均 0.22 ミリ秒で処理していることを示している．フローエントリの書き込みは新たなテナント作成時にプロアクティブに行われるため，フローエントリの書き換えにリアルタイム性が問題になることは少ないことを考慮すると，このフロー変換エンジンは十分に実用的な性能を持っていると言える．

7　考察

本研究では，マルチテナント環境において各テナントが OpenFlow 技術を自由に利用可能とすることを目指した OpenFlow ネットワーク仮想化基盤を提案した．提案手法の特徴は，各テナントネットワークを抽象化したフロースペースの競合管理と VXLAN を用いた仮想トポロジの構築による仮想技術にある．ここでは，各テナントネットワークの設計者が十分な信頼性を持ったトポロジを提示された上で，IP アドレスなどのネットワーク記述子の定義により自由にネットワーク構成を設計することができる．

既存研究の FlowVisor では各フロースペース間の競合検証は行っておらず，衝突の回避は運用者の設計に任されていた．この観点では，仮想化よりネットワークのパーティショニング技術に近いと言える．また，山中らの研究 [5] では，ネットワークのエッジで各仮想ネットワーク毎のタグ付けを特定の MAC アドレス付与することで実現しており，各テナントで記述可能なフロー定義に制限がある．加えて，SDN 仮想化基盤に関連している研究として我が国で NICT が主導している VNode[6]，米国の GENI[7]，欧州の OFELIA[8] など SDN/OpenFlow テストベッドのプロジェクトが挙げられる．これらの内，VNode と GENI ではより独立したネットワークの割り当てを目指し，OpenFlow スイッチをベースに専用ハードウェアとマネジメントモデルを実装したネットワーク仮想化ノードを用いてデータプレーンを構成している．OFELIA プロジェクトは FlowVisor をベースとしたアーキテクチャとなっており，各利用者に対して VLAN-ID を指定したフロースペースを予め割り当てることで実験用ネットワークを分割している．これら 3 つのプロジェクトでは，研究開発を支援する SDN テストベッドとしての性質から物理ネットワーク上に作成したフルメッシュの仮想トポロジをそのまま提示しており，テナントネットワークの抽象化は考慮されていない．

本研究の提案手法は各テナントネットワーク毎に自由なネットワーク定義を可能とし，その独立性を保つような制御を実現している．これにより通常の TCP/IP ネットワークの設計・構築を基本として，OpenFlow 技術により柔軟な制御を導入することができるようになるため，現状の組織の情報基盤のバックエンドをクラウド上に移すような場合においても，ネットワーク制御の柔軟性と従来並の設計の容易さの両方の利点を提供することが可能になる．

8　まとめ

本研究では，IaaS において各テナントが OpenFlow 技術を自由に利用可能なネットワーク仮想化基盤を提案した．本研究の提案手法では，マルチテナント環境において各テナントネットワーク毎に自由なネットワーク定義を可能とし，その独立性を保つような制御を実現している．これによって，IaaS 提供者は各テナントに対して OpenFlow 技術を自由に利用可能な柔軟なテナントネットワークを提供することができる．

文　献

[1] N. McKeown, ”Software-defined networking,” INFOCOM keynote talk, vol. 17, no. 2, pp. 30-32, 2009.

[2] N. McKeown et al., ”OpenFlow: enabling innovation in campus networks,” ACM SIGCOMM Computer Communication Review, vol. 38, Issue 2, pp. 69-74, April 2008.

[3] R. Sherwood et al., ”FlowVisor: A Network Virtualization Layer,” Tech. Rep. OPENFLOW-TR-2009-01, OpenFlow Consortium, October 2009.

[4] M. Mahalingam, et al., ”Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks.” Internet Engineering Task Force, RFC 7348, August 2014.

[5] 山中広明, 石井秀治, 河合栄治. “フロースペース仮想化による仮想 OpenFlow ネットワークの実現,” 電子情報通信学会技術研究報告. NS, ネットワークシステム 112.85, pp. 67-72, 2012.

[6] Y. Kanada, K. Shiraishi and A. Nakao, ”Network-virtualization nodes that support mutually independent development and evolution of node components,” 2012 IEEE International Conference on Communication Systems (ICCS), pp. 363-367, November 2012.

[7] M. Berman et al., ”GENI: A federated testbed for innovative network experiments,” Computer Networks, vol. 61, pp. 5-23, March 2014.

[8] M. Suñé et al., ”Design and implementation of the OFELIA FP7 facility: The European OpenFlow testbed,” Computer Networks, vol. 61, pp. 132-150, March 2014.