マルチテナント向けOpenFlowハイパーバイザのための フローエントリ検証手法の検討

樋口 俊[1,a)]　廣津 登志夫[1,b)]

概要：近年のサーバ仮想化技術の発展により，既存のITインフラをデータセンタ上で仮想化して提供するクラウドサービスの普及が進んでいる．こうしたサービスを提供しているマルチテナント型データセンタでは多数のテナント向けネットワークを提供するためにネットワークの仮想化や制御が必要となり，それらを実現するOpenFlow技術に注目が集まっている．既存研究のFlowVisorでは，OpenFlowネットワークを仮想化し複数のコントローラを利用可能とすることでマルチテナント環境への対応を実現しているが，個々の仮想ネットワークの制御ルールに衝突がないことが前提であった．しかし，将来的に各テナントにOpenFlowによる柔軟な制御を許すことを考慮すると，各テナントが自由に設計したネットワーク制御ルールを衝突無く処理する必要が生じる．そこで本論文では，各テナントが定義するアドレス空間の衝突管理とフローエントリの衝突検証の手法を提案することで，各テナントの仮想ネットワーク間で分離性が保証されることを目指す．

A Study of Flow-Entry Verification Methods for Multi-Tenant OpenFlow Hypervisor

Shun Higuchi[1,a)]　Toshio Hirotsu[1,b)]

1. はじめに

サーバ仮想化技術の発展によって，組織が必要とするITインフラ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で共有す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が分離されている必要がある．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の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テナントネットワーク上で自由にレイヤ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．従来に比べより動的に仮想ネットワークや仮想マシンが増減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 Network (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を可能としている．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，構成変更に伴う

[1] 法政大学
Hosei University
[a)] shun.higuchi.6j@stu.hosei.ac.jp
[b)] hirotsu@hosei.ac.jp

VLAN 管理を 1 つのコントローラに集約することができる．これによって，IaaS 提供者が SDN 技術を利用することで柔軟な仮想ネットワークの管理を実現している．またその一方で，SDN 技術によって可能となった「ソフトウェアによる柔軟なネットワーク制御機能」を，テナント側からテナントネットワークの制御にも用いたいという要求が存在していた．このように SDN 技術を IaaS 基盤の管理・運用技術として用いるのではなく，各テナントが利用可能なネットワーク制御技術として提供する場合には，各テナントが発する OpenFlow などのプロトコルによる SDN 処理の要求を直接処理できる仕組みが必要となる．

これに対して，複数の OpenFlow コントローラの要求を処理する既存研究として FlowVisor[3] が挙げられる．FlowVisor[3] では OpenFlow コントローラとスイッチ間にプロキシとして配置し，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 ネットワーク上で複数のコントローラからそれぞれ個別に OpenFlow スイッチを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従って書き込むフローエントリなどの設定を行う．この仕組みによって複数の仮想 OpenFlow ネットワークを構築し，それぞれを異なるテナントコントローラから制御することが出来ている，この手法では各フロースペース間に重なりがないことが前提となっており，マルチテナントネットワークへの適用を考えた場合には，IaaS 管理者が提供したフロースペースの範囲内でテナントネットワークを設計することになる．また，あるフロースペースにおいて他のフロースペースの監視をするようなケースではフロースペース間の重なりは想定されているが，それ以外のネットワークではフロースペースの定義を非常に注意深く行わないと意図しないトラフィック制御を引き起こしてしまう．

そこで，本研究では FlowVisor をベースとしてテナント毎の自由なネットワーク設計を可能とする OpenFlow ネットワーク仮想化手法の実現を目指す．これを実現する上で必要となるトラフィックの分離性を保証するために，フロースペースの定義の検証による衝突管理と，フローエントリの書き換えによる衝突回避による手法について提案する．具体的には，FlowVisor 上で各テナントがネットワーク設計として定義したフロースペース間において重複部分を検証・管理し，コントローラが書き込むフローエントリに対して衝突検証と衝突を回避するための書き換えを行う．本論文では，FlowVisor の仕組みと問題点について説明した上で各テナントのフロースペース間とフローエントリの検証手法について検討と提案を行う．

2. OpenFlow/SDN

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 ではネットワークの経路制御を担う OpenFlow コントローラと，その制御に従ってパケットの転送を行う OpenFlow スイッチによって，従来のネットワーク構成からコントローラプレーンとデータプレーンに分離したネットワークを一元管理可能と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 MAC アドレスや IP アドレス，トランスポート番号，VLAN-ID などのパケットに対する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，スイッチがこのフローエントリに従ってパケットを処理することで柔軟な経路制御を可能としている．また，ネットワーク構成の変更に伴いスイッチの再設定が必要な場合では変更内容をコントローラ上で記述することで全てのスイッチに変更が反映されるなど，高いネットワークの管理性を実現している．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構築される 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 ではコントローラが全てのスイッチを制御しトポロジ情報を把握しているため，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を行うことが出来る．加えて，OpenFlow を用いた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できるレイヤ 1〜4 のヘッダ情報を用いたネットワークの論理的な分割が可能となっている．アドレス空間を予め分割して利用することで，通常の VLAN-ID の上限を超えた数の仮想ネットワークを作成することが出来る．

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 ネットワークに対して複数のコントローラから個別にスイッチ制御を行う，1 つの OpenFlow ネットワークを論理的に複数の仮想 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し制御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 ネットワークを構築・制御するといった利用形態が不可能だった．

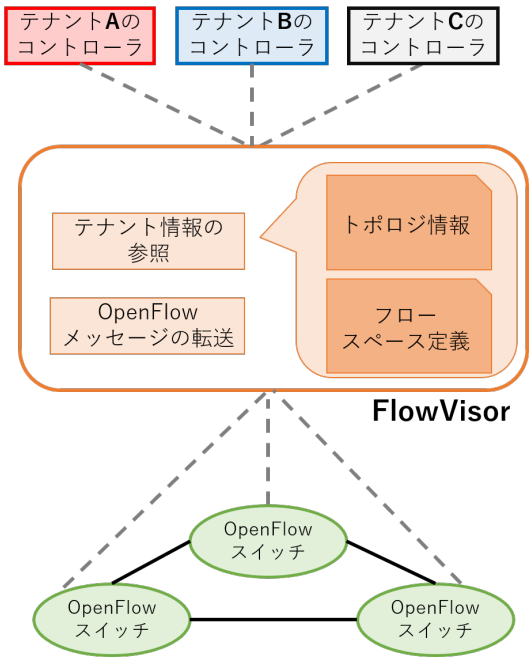

図 1　FlowVisor

3. FlowVisor

既存研究である FlowVisor では，図 1 のように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 の管理者は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，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．

3.1 フロースペース

FlowVisor では，FlowVisor 自体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．表 1 で示されるように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ことができる．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，フロースペースの衝突を検証する仕組みが用意されていないため，FlowVisor の管理者が注意深くフロースペースの定義を行う必要がある．

3.2 FlowVisor の動作

FlowVisor は OpenFlow チャンネル上で透過型プロキシとして機能するため，その動作として透過的な OpenFlow メッセージの転送制御を行う．複数台のコントローラとスイッチ間で透過的な転送制御を行うため，Packet In や Port Status メッセージなどのスイッチからコントローラへメッセージを転送する場合と，Flow Mod メッセージのようなコントローラからスイッチへメッセージを転送する場合とで動作が異なっている．

まず，スイッチからコントローラへメッセージを転送する場合を説明する．この場合ではそのメッセージがどのコントローラに影響を及ぼすのかを特定した上で転送を行う必要がある．例としてスイッチの物理ポート状態が変化したことを通知する Port Status メッセージでは，その物理ポートを利用している全てのテナントコントローラに影響が及ぶため，FlowVisor が保持している各テナントネットワークのトポロジ情報から対象となる全コントローラを検索しそれら全てに対して Port Status メッセージを転送する．また，Packet In メッセージの場合にはパケットがどのテナントに属するものなのかを特定するためにフロースペース定義に対して検索を行い，該当したフロースペースのテナントコントローラへ向けてメッセージを転送する．

次に，コントローラからスイッチへのメッセージ転送について説明する．この場合にはどのメッセージでも同じ動作として，FlowVisor が持つテナントネットワークのトポロジ情報を参照して対象となるスイッチへの転送が行われる．この時，トポロジ情報上で自身のテナントに属していないスイッチへのメッセージを送信することはできず，送信した場合にはメッセージの転送エラーとしてエラーメッセージがコントローラに返される．

3.3 FlowVisor の問題点

FlowVisor では，各テナントネットワークに割り当てられているフロースペースがそれぞれ独立で，テナントコントローラはそのフロースペースの範囲内でフローエントリを設定するということが前提となっているため，意図しない内容のフロースペースや誤った設定のものが定義された場合には正常ではないネットワーク制御が実行される可能性がある．マルチテナントネットワークの提供形態として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た場合には，それぞれが定義したフロースペースを FlowVisor でそのまま利用すると，他テナントのフロースペースと衝突するようなフローエントリを書き込まれることで意図しないトラフィック制御が行われるなどの問題が発生する．このことから，このような想定のマルチテナントネットワークではフロースペースとフローエントリの衝突を検証する仕組みの実装が必要となる．

フローエントリが衝突してしまう例として，表 1 のテナ

表 1　フロースペースの例

Slice	DPID	Priority	VLAN	Src MAC	Dst MAC	Src IP	Dst IP	Src TCP	Dst TCP
Tenant A	1	100	50	*	*	*	*	80, 22	*
Tenant B	1	100	50	*	*	10.0.1.0/24	*	80	*
Tenant C	1	100	50	*	*	10.0.2.0/24	*	80	*

ント A のコントローラにおいて接続している DPID = 1 のスイッチに *Src TCP = 22, action = DROP* という SSH のセッションを禁止するフローエントリを書き込んでしまった場合が存在する．表 1 ではテナント A の src TCP 以外のマッチフィールドが* つまりワイルドカードとなっているため，この値についてテナントユーザは自由に利用できる．しかし，前述のようなフローエントリを書き込んだ場合には DPID = 1 のスイッチを通過し送信元 TCP ポートが 22 番のすべてのパケットに適用され，該当パケットが全て破棄されるので他のテナントネットワークでも SSH の接続が不可になってしまう．また，この例では本来の意図とは異なってパケットが破棄されているが，OpenFlow ではパケットへのアクションとしてヘッダ情報を書き換えて転送することが可能であるため，許可されていない他テナントのトラフィックを自分のテナントネットワーク上のサーバに転送して盗聴するなどの行為も可能となってしまう．

このように，FlowVisor で各テナントが自由なネットワーク設計とフロースペース定義を行った場合には，重複部分があるフロースペースによって意図しない挙動を起こすフローエントリが書き込まれる．これは，フローエントリにおいてワイルドカードのような値を L1 から L4 までのマッチフィールドで柔軟に設定できる OpenFlow の仕組みによるものである．前述の例のように，送信元 TCP ポート番号以外がワイルドカードになっているフローエントリがそのままスイッチに書き込めてしまうため，割り当てられていないフロースペースのトラフィックに対しても制御が出来てしまっている．

4. フローエントリ検証に基づく仮想化

本研究では，既存研究である FlowVisor をベースとして各テナントに割り振られるフロースペースにおいて重複を検証・管理した上で，各テナントコントローラが書き込むフローエントリの衝突検証と書き換えを行うことでテナント毎の自由なネットワーク設計を可能とする OpenFlow ネットワークの仮想化手法を提案する．図 2 に示したように本手法で提案する検証機構を FlowVisor に追加実装する．本手法では，まず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・管理する機構を導入する．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てしまうことを可能な限り回避することが出来る．加えて，定義内でアドレス空間が重複しているフロースペースにおいてフローエントリが書き込まれる場合には，フローエントリの衝突検証とマッチフィールドの書き換えを行うことで各テナント間のトラフィックの分離性を保証する．このように，フロースペースに対する検証・管理を予め行っておくことで，フローエントリに対する書き換え処理を最低限に抑えることが出来る．

また，これらを実現するために本手法では既存研究の FlowVisor で定められていたフロースペースの定義に変更を加え，各テナントから利用できるマッチフィールドの要素と組み合わせに対して一定の制限を設ける．従来のフロースペースでは，OpenFlow のマッチフィールド全ての要素を対象として任意のフィールドを利用することが出来ていた．しかし，この方法ではワイルドカードを利用することで意図しないトラフィックにもマッチするフローエントリを書き込むことが出来てしまう．これに対して本手法では，予め指定された範囲のアドレス空間から構成される組み合わせのマッチフィールドに利用を制限する．これによって定義されたネットワーク内のみに制御を限定すると同時に，実用上ネットワークの定義では値が指定されないフィールドに対する検証を行う必要がなくなる．

4.1 フロースペースの定義

本研究で扱うフロースペースの定義について説明する．このフロースペースは 3.1 節の FlowVisor のものから変更されて独自の定義になっている．既存研究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，本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 2 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，Src/Dst TCP ポートという L2〜L4 の 5 種類のヘッダ情報を指定することができる．これらの定義は図 3 で示すように JSON 形式で記述する．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

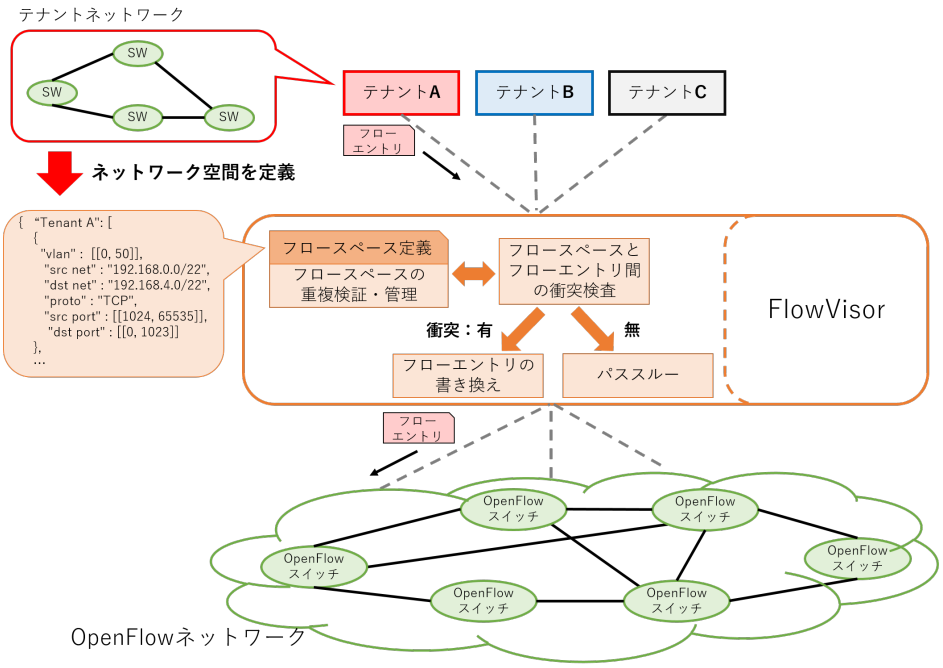

図 2　提案手法の概要

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．図 3 の定義例をまとめた表 2 において，最下段にある定義例 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, Src TCP = 80
- VLAN ID = 101, Src IP = 192.168.64.0/20, Dst IP = 192.168.64.0/20, Src TCP = 80

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．また，表 2 の上段にマッチフィールド中で利用可能なアドレス空間を示しているが，その内の VLAN ID の上限が本来の上限である 4096 個の半分となっている．これはフロースペースの重複管理において，衝突を解決するために必要な独立したアドレス空間を予め管理用空間として確保し，必要な場合に適時フロースペースに割り振るためである．

4.2　フロースペースの重複検証とフローエントリ

前節で説明した手法に基づいて定義したフロースペースにおいて，フロースペース間の重複検証とフローエントリの衝突について説明する．3 つのテナント A，B，C に対して定義されたフロースペースの例を示した表 3 では，最上段のテナント A の定義が以下のテナント B・C のフロースペースを完全に包含してしまっているため，このフロースペース A は B・C との重複が存在する独立ではない定義と

```
{
  "定義例１" : [
    {
      "vlan" : [[0, 50]],
      "src net" : "192.168.0.0/22",
      "dst net" : "192.168.4.0/22",
      "proto" : "TCP",
      "src port" : [[1024, 65535]],
      "dst port" : [[0, 1023]]
    },
    {
      "vlan" : [[0, 50]],
      "src net" : "192.168.4.0/22",
      "dst net" : "192.168.0.0/22",
      "proto" : "TCP",
      "src port" : [[0, 1023]],
      "dst port" : [[1024, 65535]]
    }
  ],

  "定義例２" : [
    {
      "vlan" : [[100], [101]]
      "src net" : "192.168.64.0/20",
      "dst net" : "192.168.64.0/20",
      "proto" : "TCP",
      "src port" : [[80]],
      "dst port" : [[80]]
    }
  ]
}
```

図 3　フロースペースの定義例

表 2　フロースペースの上限と定義例

Rule ID	Space Name	VLAN	Src IP	Dst IP	Src TCP	Dst TCP
1	利用上限	0〜2047	0.0.0.0〜255.255.255.255	0.0.0.0〜255.255.255.255	0〜65535	0〜65535
2	定義例 1	0〜50	192.168.0.0/22	192.168.4.0/22	1024〜65535	0〜1023
3	定義例 1	0〜50	192.168.4.0/22	192.168.0.0/22	0〜1023	1024〜65535
4	定義例 2	100, 101	192.168.64.0/20	192.168.64.0/20	80	*

なっている．これに対して互いが独立なフロースペースとなっている定義では，表 3 中のテナント B と C のフロースペースでは指定されている Src IP の値のようにマッチフィールド内の何れか 1 つ以上にそれぞれ独立な範囲の値が指定されている．本手法で提案したフロースペースの定義では，指定した範囲のアドレス空間とそれらの AND を取った組み合わせのみが利用可能なマッチフィールドとなるため，それぞれの組み合わせに対して包含関係の検証を行うことで重複を検出する．

完全な包含関係を持つフロースペースでは，重複を管理した上で書き込むフローエントリの衝突を検出・回避することが必要となる．表 3 のようにフロースペースが定義された場合，テナント A のコントローラでは表 4 のフローエントリが書き込まれることで他のテナントのフローエントリと衝突する可能性がある．表 4 ではその例として他のフロースペース定義と衝突するフローエントリと衝突しないものの 2 種類を示している．まず，上段のフローエントリの例ではテナント A のフロースペースでワイルドカードになっている Src IP の値が他のテナント B・C のフロースペースで指定された値の範囲を包含しているため，それらのテナントのフローエントリと衝突した上で他テナントのトラフィックがこのフローエントリによって制御されてしまう．これに対して，下段の衝突しないフローエントリの例では，マッチフィールドに Src IP=10.0.0.1 という他のテナントのアドレス空間と独立な値が指定されているためフローエントリの衝突が起きない．このように，各フロースペースのマッチフィールドに指定されている値の包含関係について検証し，他のテナントの値を包含している場合には空いている独立なアドレス空間から新たに値を抜き出し，フローエントリに指定することで衝突の検証と回避が可能となっている．

5. フローエントリの衝突検証

4 節で述べたようなフローエントリの衝突を回避するために，OpenFlow スイッチに書き込むフローエントリに対して FlowVisor 上で実行する 2 段階の検証手法を提案する．まず，書き込むフローエントリのマッチフィールドと自身のフロースペースで定義されているネットワーク空間の整合性検査を行う．次に，フローエントリ中のマッチフィールドについて他のテナントのフロースペースに定義されている値を包含しているワイルドカードの部分を空いている独立した値へ自動的に展開する．これらによって，フロースペースの定義と異なるフローエントリを禁止しながらワイルドカードを利用したフローエントリによる衝突を検出・回避し，テナントネットワーク間でトラフィック制御が分離されていることを保証する．

5.1 フロースペースとの整合性検査

まず，テナントコントローラから FlowVisor に送信される Flow Mod メッセージ中のフローエントリについて，テナントのフロースペース定義から外れたマッチフィールドが設定されていないかという整合性検査を行う．整合性検査では単純にフローエントリ中のマッチフィールドにおいてワイルドカードではない値について，フロースペースで定義されている値の範囲を超えていないかというアドレス空間の大小比較を行う．この手順の時点で，フロースペース定義と異なる値のフローエントリを書き込もうとしていた場合には，Flow Mod メッセージを破棄してコントローラに Flow Mod メッセージの送信エラーを通知する．

5.2 ワイルドカード部分の展開

5.1 節の整合性検査を通過したフローエントリについて，マッチフィールドのワイルドカード部分が他のフロースペースで定義されているアドレス空間と衝突しないようにフローエントリの書き換えを行う．ここでは，3.3 節と同じく表 1 のテナント A のコントローラにおいて，表 5 の上段で示されているような「送信元 TCP ポート番号が 22 番のパケットは全て破棄する」というフローエントリを書き込みしようとした場合を例に動作を説明する．この場合，まず送信元 TCP ポートの値はフロースペース中に収まっているので 5.1 節の整合性検査を通過する．次に，このフローエントリの送信元 TCP ポート以外のマッチフィールドは全てワイルドカードになっているが，表 1 のフロースペースの定義から VLAN-ID と送信元 IP アドレスについてテナント B のフロースペースを包含し，送信元 IP アドレスと宛先 IP アドレスの空間についてテナント C のフロースペースを包含していることがわかるため，これらについてそれぞれ 1 つ以上独立な値を指定することでフローエントリの衝突を回避する．それぞれ空いているアドレス空間を用いてフローエントリの書き換えを行った結果が表 5 の下段となっている．このように，ワイルドカード部分を書き換えることでマッチフィールドが衝突しないフローエン

表 3　フロースペースの重複例

Rule ID	Space Name	VLAN	Src IP	Dst IP	Src TCP	Dst TCP
1	Tenant A	50	*	*	80, 22	*
2	Tenant B	50	10.0.1.0/24	*	80	*
3	Tenant C	50	10.0.2.0/24	*	80	*

表 4　表 3 中のテナント A におけるフローエントリの例

Entry	Match Field	Action
衝突する フローエントリ	VLAN ID = 50 Src TCP = 80	Output: port 2
衝突しない フローエントリ	VLAN ID = 50 Src IP = 10.0.0.1 Src TCP = 80	Output: port 2

表 5　ワイルドカード部分の書き換え

Entry	Match Field	Action
書き換え前 フローエントリ	Src TCP = 80	DROP
書き換え後 フローエントリ	Src TCP = 80 Src IP = 10.0.0.0/24 VLAN ID = 2048	DROP

トリを Flow Mod メッセージとして転送し，フローエントリが他のテナントネットワーク上のトラフィックを誤って制御しないことを保証する．

6. 実装

本提案手法の核になるフロースペースの衝突検証システムを構成する「フロースペースマネージャ」と「フロー変換エンジン」のうち，前者のプロトタイプシステムの実装を行った．本節ではその実装と初期的な性能評価について述べる．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，図 3 のように記述された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 Ruby2.3 で実装し，初期的な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 2.8GHz, 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．フロー変換エンジンは現在実装中であるが，基本的にはあるフロースペースの定義のみを検査すれば良いため，フロースペースマネジャよりは探索対象のフロースペース定義は少なくてすむ．

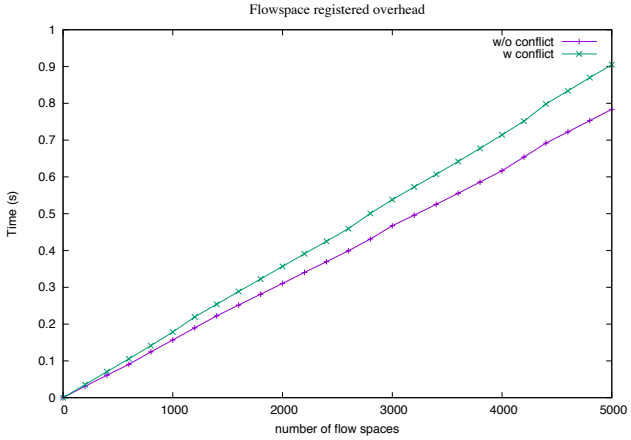

図 4　フロー登録の速度

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手法を提案した．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各フローエントリの衝突検証を用いた，OpenFlow ネットワークの仮想化手法にある．ここでは，各テナントネットワークの設計者が，IP アドレスなどのネットワーク記述フィールドの定義により自由にネットワーク構成を設計することに特徴がある．

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，仮想化というよりネットワークのパーティショニング技術であると見るのが妥当と考えられる．Sköldström[4] らの研究では，この FlowVisor を広域ネットワークの中継網に活用するための仮想化手法を提案

しているが，資源管理などを視野に入れており MPLS などの下位層へのネットワーク分離技術へのマッピングを主としている．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．

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，フロースペースによる仮想ネットワークの定義に対する検証をベースに置いたネットワーク仮想化技術について提案した．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 技術を自由に用いることができる柔軟なテナントネットワークを提供することが可能になる．また，フロースペース管理のプロトタイプの初期評価においては，十分に実現可能な性能があると見込まれている．今後は，フロー書き換えエンジンの実装を進め，FlowVisor と連携して稼働する仮想化技術を完成させる予定である．

謝辞 本研究は JSPS 科研費 JP15K00138 の助成を受けたものです．

参考文献

[1] McKeown, Nick. ”Software-defined networking.” INFOCOM keynote talk 17.2 (2009): 30-32.

[2] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Turner, J.: *OpenFlow: enabling innovation in campus networks* ACM SIGCOMM Computer Communication Review Volume 38 Issue 2, (April 2008): 69-74.

[3] Sherwood, R., Gibb, G., Yap, K.-K., Appenzeller, G., Casado, M., McKeown, N., and Parulkar, G.: *FlowVisor: A Network Virtualization Layer*, Tech. Rep. OPENFLOW-TR-2009-01, OpenFlow Consortium, (October 2009)

[4] Sköldström, P., Yedavalli, K. *Network virtualization and resource allocation in openflow-based wide area networks*, In 2012 IEEE International Conference on Communications (ICC), (2012, June): 6622-6626.

[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 (2012): 67-72.