OpenFlowを用いた不正通信制御における コントローラ冗長化手法の提案

下川 大貴[1]　池部 実[2]　吉田 和幸[3]

概要：これまで学内で開発・運用している不正通信検知システムが検知した攻撃者の通信を OpenFlow を用いて制御する手法を提案してきた．提案手法の本運用に向けて，実際のネットワークにおいて収集したパケットデータを用いて，実環境を再現した場合，OpenFlow による通信制御を実施しても評価用トラフィックに与える影響が小さいことが確認できた．一方，OpenFlow を用いた不正通信制御システムの本運用を考慮した場合，OpenFlow コントローラが単一障害点となる．起こりうる問題として，OpenFlow コントローラと OpenFlow スイッチ間の接続が失われた場合，OpenFlow スイッチ内のフローエントリが消失し，OpenFlow ネットワーク全体が停止する可能性がある．その結果，不正通信の制御も不可能となる．しかし，OpenFlow プロトコルには OpenFlow コントローラの冗長化については明確な規定はないため独自の冗長化手法を実装する必要がある．そこで本稿では不正通信制御における OpenFlow コントローラの冗長化について提案する．OpenFlow コントローラに起こりうる障害を想定し，冗長手法によって OpenFlow ネットワークが正常に機能しているか動作を検証をした．さらに，OpenFlow コントローラ間でやりとりする死活管理メッセージの送信間隔とコントローラ切替時間の適切な送信間隔，切り替え判断タイミングを考察した結果を報告する．

キーワード：ネットワーク, OpenFlow, 不正通信, 通信制御, コントローラ冗長化

Proposal of Controller Redundancy for controlling malicious communication using the OpenFlow

Shimokawa Daiki[1]　Ikebe Minoru[2]　Yoshida Kazuyuki[3]

Abstract: In previous work, we have been developing an anomaly communication detection system. And, we proposed a malicious communication controller method using the OpenFlow by the results of the anomaly communication detection system. We was experimented the proposed system in the reproduction environment of the actual network. As a result, we have confirmed that evaluation traffic is not affected by our proposed method. On the other hand, openflow contorller becomes the single point of failure in our proposed system. If openflow controller failure has occurred, openflow network stops and our proposed system becomes impossible the control of malicious communications. We consider to need redundancy of openflow controller is indispensable for openflow network. However, redundancy methods of openflow controllers have not defined in openflow specification. Therefore, we propose a redundancy method of openflow controllers in malicious communication control method. And, we designed and implemented a new redundancy method. We verify our redundancy method in several troubles. Moreover, we discuss transmission intervals of life-and-death messages. As a result, we confirmed the validity of our redundancy method of openflow controllers in malicious communication control method.

Keywords: Network, OpenFlow, malicious communication, communication control,controller redundancy

1. はじめに

ネットワークは社会的基盤の一つとして生活に不可欠な存在になっている．インターネットを通じて，多様なサービスが利用されている一方で，サーバの不正アクセスなど，さまざまな脅威が存在する．警察庁の平成 26 年中のインターネット観測結果 [1] によると，シグネチャを用いて検知した不正侵入等の行為の件数は，1 日・1IP アドレス当たり 177.1 件で，平成 25 年時 の調査と比較して不正な通信が約 10 倍に増加している．

我々は，これまでにインターネットから大分大学のネットワークに対する不正な通信を検知するため，TCP スリーウェイハンドシェイクの手順と異なるパケットを送信した攻撃者を検知し，遮断する” 不正通信検知システム”[2] を開発・運用してきた．従来，不正通信検知システムにより検知した攻撃者は，スイッチの ACL(Access Control List) を用いて遮断していた．しかし，ACL では攻撃者の登録件数に上限があり，攻撃者を 1 件ずつ登録するため複数の攻撃者を登録するために時間がかかる問題があった．そのため，攻撃者の ACL による遮断時の問題を改善するため，不正通信検知システムが検知した攻撃者の通信を OpenFlow を用いて制御する手法を提案してきた [3][4]．これまでに仮想的な OpenFlow ネットワーク環境を構築し，iperf により発生させたトラフィックを DoS や DDoS 攻撃と見立てて OpenFlow を用いた不正通信制御システムが攻撃者を遮断する際の評価用トラフィック (ping や iperf) への影響を検証してきた．さらに，実際のネットワークで収集したパケットデータを用いて不正通信検知システムが検知した大量の攻撃者の送信元 IP アドレスを一度に投入・削除した際の性能を検証した．これらの実験により，実際のネットワーク環境で発生する攻撃を想定した状況においても攻撃者登録処理が評価用のトラフィックに与える影響が小さいことを確認した．

しかし，実際のネットワーク環境での OpenFlow を用いた不正通信制御システムの運用を考慮した場合，OpenFlow コントローラに障害が発生して，OpenFlow スイッチとの接続がすると，OpenFlow スイッチに存在しているフローエントリの消失など，OpenFlow ネットワークに多大な影響が生じることがある．そこで，本論文では現在の不正通信検知システムと OpenFlow を用いた通信制御システムの構成を考慮した OpenFlow コントローラの冗長化手法を提案し，様々な障害を想定した実験にて，提案システムによる OpenFlow コントローラの冗長化手法を評価する．

第 2 章では，我々がこれまでに提案してきた不正通信検知システムおよび OpenFlow を用いた不正通信制御システムについて説明する．第 3 章では，不正通信検知システムと OpenFlow を用いた通信制御システムにおける構成を考慮した OpenFlow コントローラの冗長化手法について説明する．第 4 章では，OpenFlow コントローラに起こりうる障害を想定し，冗長手法によって OpenFlow ネットワークが正常に機能しているかの動作検証，OpenFlow コントローラ間での送受信する死活管理メッセージの送信間隔とコントローラ切替り替え判断タイミングの適切な時間を考察した結果を報告する．第 5 章では，本稿のまとめと今後の課題を述べる．

2. OpenFlow を用いた不正通信制御システム

本章では不正通信検知システムから OpenFlow コントローラに対して不正通信の送信元に関する情報を通達し，OpenFlow コントローラは OpenFlow スイッチに対して動的にフローエントリを設定し，攻撃者の通信を遮断する手法について述べる．

2.1 不正通信検知システム

我々が開発・運用している不正通信検知システム [2] はインターネットから学内ネットワークへ送信される TCP パケットのフラグやコネクションの接続状態に注目して不正通信を検知する．

2.2 OpenFlow を用いた不正通信制御システムの概要

不正通信検知システムが検知した攻撃者情報を用いて OpenFlow で通信を制御する手法について述べる．

2.2.1 OpenFlow

OpenFlow[5] は従来のネットワーク機能における機能をデータプレーンとコントロールプレーンに分けることで，柔軟に通信を制御できる．コントロールプレーンである OpenFlow コントローラ，データプレーンである OpenFlow スイッチはセキュアチャネルと呼ばれる専用線を介して制御メッセージをやりとりする．コントロールプレーンは経路制御方法や受信パケットの扱い方を指示し，複数の OpenFlow スイッチを集中管理する．データプレーンは OpenFlow コントローラから設定されたフローエントリにもとづき，パケットの転送や破棄などを実行する．また，フローエントリは処理を対象のパケットを特定する「マッチングルール」，パケットに対する処理内容である「アクション」，フローエントリの「優先度」，フローエントリごとの参照回数などの「統計情報」，以上の要素から構成されている．

[1] 大分大学大学院工学研究科知能情報システム工学専攻
Course of Computer Science and Intelligent Systems, Graduate School of Engineering, Oita University
[2] 大分大学工学部知能情報システム工学科
Department of Computer Science and Intelligent Systems, Faculty of Engineering, Oita University
[3] 大分大学学術情報拠点情報基盤センター
Center for Academic Information and Library Services, Oita University

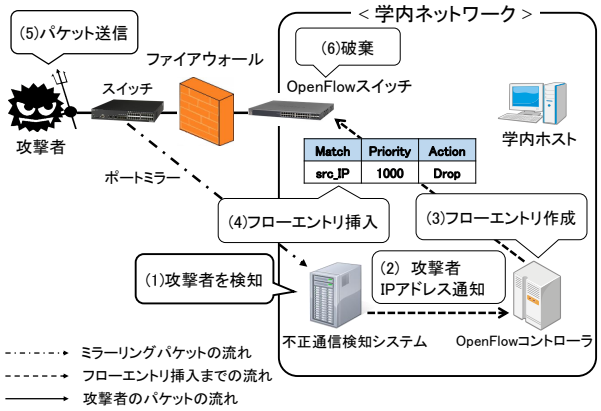

図 1　OpenFlow を用いた不正通信制御システムの構成図と制御手順

2.2.2　OpenFlow による不正通信の制御手法

OpenFlow を用いた不正通信の制御手順および，システム構成を図 1 に示す．OpenFlow を用いた不正通信の制御は以下の手順で実施する (図 1).

(1) 不正通信検知システムが攻撃者を検知
(2) 不正通信検知システムは OpenFlow コントローラへ攻撃者 IP アドレスを通知
(3) OpenFlow コントローラは通知された攻撃者 IP アドレスをもとにフローエントリを作成
(4) OpenFlow コントローラは OpenFlow スイッチに対して FlowMod メッセージにより，フローエントリを設定
(5) 攻撃者がパケットを送信
(6) 攻撃者のパケットは OpenFlow スイッチでフローエントリのアクションに従い，破棄

2.2.3　運用上の問題点

実際に，図 1 に示したように，学外と学内の境界に設置した OpenFlow スイッチにて OpenFlow を用いた不正通信制御システムを運用しようとした場合，OpenFlow スイッチは容易に冗長化することができるが，OpenFlow コントローラは単純な冗長化ができず，単一障害点となる．OpenFlow コントローラと OpenFlow スイッチ間のセキュアチャネルが何らかの理由で切断した場合，最悪のケースでは，OpenFlow スイッチ内のフローエントリが消失することがある．そのため，OpenFlow コントローラに障害が起きた場合，不正通信の制御はもちろん，OpenFlow により制御しているネットワーク全体が停止する可能性がある．このような問題を発生させないために，OpenFlow version 1.2 では複数台の OpenFlow コントローラを 1 台の OpenFlow スイッチに接続できる仕様が新たに追加された．しかしながら，複数台存在する OpenFlow コントローラ間の死活管理や役割管理などは特に規定されておらず，これらの機能をどのように利用すれば OpenFlow コントローラの可用性が向上するかは，それぞれの OpenFlow コントローラ側で考慮しなければならない．

3.　不正通信制御における OpenFlow コントローラの冗長化手法の提案

本論文では 2.2.3 項で述べた問題を解決するために，不正通信制御における OpenFlow コントローラの冗長化手法について提案する．

3.1　OpenFlow を用いた不正通信制御における要求要件

図 1 に示したシステム構成において，OpenFlow を用いた不正通信制御における要求要件を述べる．

- **要求 1**　OpenFlow ネットワークの停止を防ぐ
  図 1 のシステム構成では OpenFlow コントローラと OpenFlow スイッチの接続性喪失時に OpenFlow スイッチ内のフローエントリが消失し，OpenFlow で制御しているネットワークがダウンする可能性がある．図 1 では OpenFlow スイッチが学外と学内の境界に位置するため，学外との通信が不可能となる．このことから，OpenFlow ネットワークダウンを防ぐ対処が必要となる．
- **要求 2**　不正通信の制御機能の停止を防ぐ
  図 1 のシステム構成では OpenFlow コントローラと不正通信制御システム間の通信が途絶えると，不正通信の制御機能が停止する可能性があることから，不正通信制御の停止を防ぐ対処が必要となる．

3.2　冗長化の設計方針

3.1 節で記載した要求を満たすため，OpenFlow コントローラに障害が発生したとしても，OpenFlow スイッチが保有している既存のフローエントリを引き継ぎつつ，不正通信制御の継続を可能となるよう，5 つの設計方針を考えた．また，冗長化にあたってのネットワーク構成として以下の二通りが考えられ，それぞれの構成に対応した冗長化手法をとる必要がある．

- OpenFlow コントローラ群がセキュアチャネルを介して通信する場合
- OpenFlow コントローラ群がセキュアチャネルとは別のネットワークを介して通信する場合

本論文では，前者の OpenFlow コントローラ同士が OpenFlow スイッチを介して通信する場合の冗長構成を対象とする．

3.2.1　要求 1 を満たすための対処方法

(1)n 台の OpenFlow コントローラによるアクティブ/スタンバイ方式

OpenFlow スイッチに OpenFlow コントローラを n 台 (n≥2) 接続し，1 台はアクティブにし，アクティブ状態のコントローラの障害に備えて n-1 台をスタンバイの状態にする方式をとる．この方式ではアクティブ状態のコントローラに障害が発生して処理の続行が不可能になると直ちにス

表 1　Multiple Controllers における属性

属性	特徴
EQUAL	OpenFlow スイッチに対してフルアクセス権を持つ．スイッチ 1 台に対して EQUAL は複数接続可能
MASTER	EQUAL と同様の権限を持つ．スイッチ 1 台に対して MASTER は 1 台のみ接続可能
SLAVE	OpenFlow スイッチに対して read-only のアクセス権を持つ．スイッチ 1 台に対して SLAVE は複数接続可能

タンバイ状態のコントローラが稼動して処理を引き継ぐ．本冗長化手法では，アクティブ/スタンバイ方式をとるための役割を OpenFlow Switch Specification version1.2[6] より規定されている「Multiple Controllers」機能により実現する．

(2)OpenFlow コントローラの役割管理

3.2.1 節の (1) で示した，Multiple Controllers 機能により各コントローラに付与できる役割の管理方法について述べる．Multiple Controllers により定義されているコントローラ属性を表 1 に示す．

本冗長化手法では，Master，Slave のコントローラ属性を用いる．また，コントローラ属性の一つである EQUAL を用いない理由として，EQUAL 属性は OpenFlow スイッチに対してフルアクセス権を持つため，OpenFlow スイッチに接続されているすべての OpenFlow コントローラは攻撃者遮断のフローエントリを挿入することが可能である．そのため，必要以上に攻撃者遮断登録処理を行ってしまうことや OpenFlow スイッチ内のフローエントリと OpenFlow コントローラ側の攻撃者情報の整合性が取れなくなる可能性がある．3.2.1 節の (1) で示した，アクティブ系を Master，スタンバイ系を Slave とし，役割管理をする．Master に障害が発生し，処理の続行が不可能になるとスタンバイ状態である Slave が Master へと昇格し，処理を引き継ぐ．以上のように役割管理をする．

(3)OpenFlow コントローラの死活管理

OpenFlow プロトコルでは OpenFlow コントローラ間の通信は規定されていないため，独自に実装する必要がある．死活監視の方法例として，黒木ら [10][11] は，ping による疎通確認や SNMP により取得した CPU とメモリの使用率を用いて死活確認している．ping や SNMP による監視はネットワークの切断は検知できるが，OpenFlow コントローラのプロセスダウンまでは検知できない．そのため，OpenFlow プロセスダウンを検知可能な死活管理方法を考える必要がある．

3.2.2　要求 2 を満たすための対処方法

(1) 不正通信検知システムから OpenFlow コントローラへの攻撃者通知方法

従来の手法では，OpenFlow コントローラと不正通信検知システム間の TCP 通信を用いた同期式通信により，攻撃者情報を通知していた．この通知方法では OpenFlow コントローラが複数台になった場合，すべてのコントローラと不正通信検知システム間にコネクションを張り続けることになる．そこで，コントローラが複数台ある構成を考慮し，非同期式通知方法に変更する．

(2)OpenFlow コントローラ障害後の OpenFlow スイッチ内のフローエントリと不正通信検知システムが検知した攻撃者情報との整合性

本冗長化手法では，OpenFlow コントローラ (Master)(以下，Master) に障害が生じた場合，OpenFlow コントローラ (Slave)(以下，Slave) から Master へ昇格するまでの間は OpenFlow スイッチに対して攻撃者遮断のフローエントリが登録できない．そのため，遮断すべき攻撃者の情報を OpenFlow スイッチに設定することができないため，OpenFlow スイッチ内の攻撃者遮断に関するフローエントリと不正通信検知システムが検知した攻撃者情報の整合性がとれなくなる．そこで，攻撃者遮断登録に関する信頼性を確保するため，Slave が Master へ切り替わる間に OpenFlow スイッチに設定できない攻撃者情報を再度 OpenFlow スイッチに設定する仕組みが必要となってくる．

3.3　不正通信制御における OpenFlow コントローラ冗長化の実現方法

3.2 節に示した OpenFlow コントローラ冗長化における設計方針の実現方法を述べる．

(1)n 台の OpenFlow コントローラによるアクティブ/スタンバイ方式

アクティブ/スタンバイ方式による冗長化を実現するための属性管理を 3.2.1 節の (1) で述べた Multiple Controllers 機能を利用する．また，Multiple Controllers 機能をサポートしているコントローラフレームワークは，Ryu[7]，Floodlight[8]，OpenDaylight[9] などがあげられる．本研究では，Ryu を用いて OpenFlow コントローラの作成をする．

(2) 障害発生時の OpenFlow コントローラ切り替え方法

ping や SNMP による死活管理方法では不可能であったコントローラプロセスのダウンを検知するため，OpenFlow コントローラに独自に考えた死活管理を組み込んだ．メッセージの概要を表 2 に示す．また，メッセージに含まれる情報は，メッセージの種類，コントローラの IP アドレス，コントローラ属性 (Role)，コントローラ優先度 (Priority) である．Update メッセージに含まれる情報は，自コントローラ情報に加えて，他のコントローラ情報も含む．

上記のメッセージフォーマットに従いマルチキャストによりコントローラ群に送信する．3.2.1 節で示した設計方針の (2) と (3) にもとづき，Master にて障害が発生した場合のコントローラ切り替え手順および，復帰手順を図 2 に

表 2　監視メッセージ概要

メッセージの種類	内容
Join	OpenFlow コントローラ起動通知
Alive	OpenFlow コントローラ生存通知
Update	OpenFlow コントローラ情報 (コントローラ属性，IP アドレス) の更新通知
Leave	メンテナンス時などのグループ離脱通知

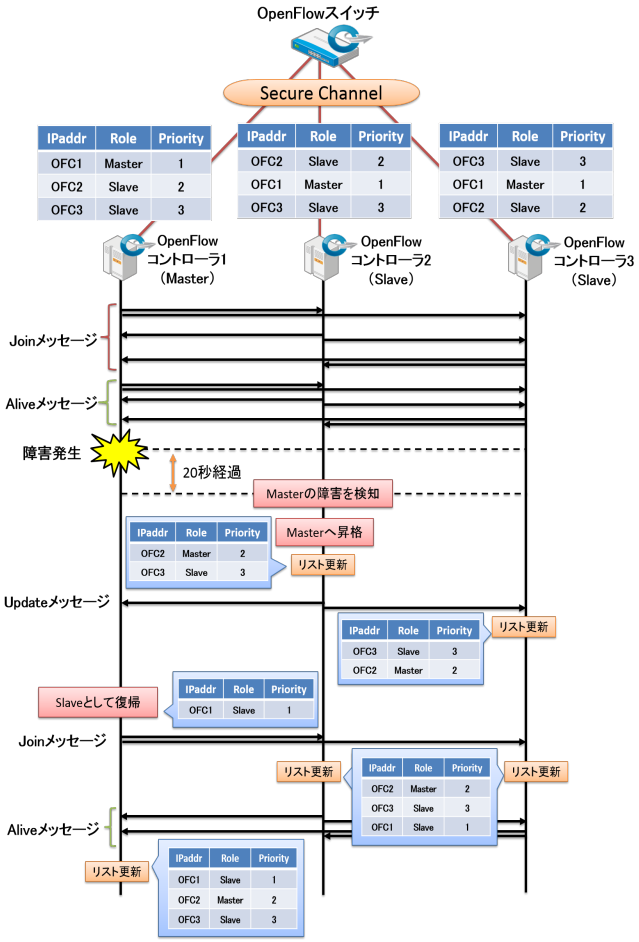

図 2　障害発生時における OpenFlow コントローラの切り替え手順および復帰手順

示す．

図 2 より，各 OpenFlow コントローラは起動時に Join メッセージを OpenFlow コントローラ群へ送信し，コントローラ属性，IP アドレスの情報を保持しておく．Alive メッセージは 5 秒間隔で送信する．また，Alive メッセージが 4 回 (20 秒以上) 届かない場合に障害とみなす．これをトリガーとして，各 Slave コントローラの優先度が最上位の Slave コントローラは自身のコントローラ属性を Master へと昇格し，OpenFlow コントローラ群へと Update メッセージを通知する．また，OpenFlow コントローラのコントローラ優先度はあらかじめ管理者が任意の優先度を各コントローラへ設定する．

OpenFlow コントローラの復帰手順として，コントローラ群へ復帰するコントローラは役割を Slave へと降格させ，Join メッセージを送信することでコントローラ群へ復帰したことを通知する．ダウンしたコントローラの役割が Master の場合，復帰時の役割が Master のままであると現在の Master と役割が競合するため，復帰時に Slave へと変更をする．Slave への変更は管理者がコントローラ復帰時に行うことを想定している．コントローラ群の Join メッセージを受信し，リストを更新したコントローラは復帰したコントローラの死活管理を再開する．

以上の手順にて，Master に障害が発生した場合のコントローラ切り替えおよび，ダウンしたコントローラの復帰をする．

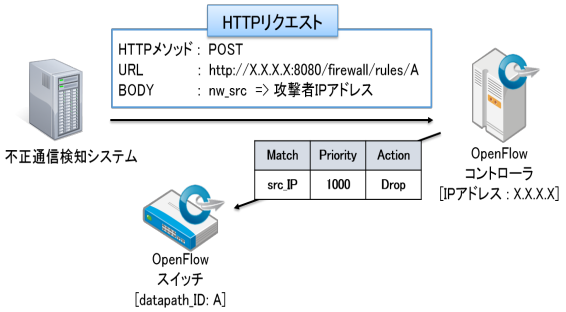

図 3　REST API による攻撃者情報の通知

(3) 不正通信検知システムから OpenFlow コントローラへの攻撃者通知方法

複数の OpenFlow コントローラに攻撃者情報を通知するため，非同期式通信による方法を実装する．そこで OpenFlow コントローラの攻撃者遮断登録処理を REST API を用いることにより，不正通信検知システムから攻撃者情報を含む HTTP リクエストを OpenFlow コントローラへ送信することで OpenFlow スイッチへ攻撃者遮断登録が可能となる．図 3 に不正通信検知システムから攻撃者情報を受け取り，攻撃者遮断登録までの挙動を示す．図 3 に示した HTTP リクエストを送信することで，OpenFlow コントローラはリクエストに含まれる攻撃者 IP アドレスを遮断するフローエントリを作成し，OpenFlow スイッチへ設定する．

(4)OpenFlow コントローラ障害後の OpenFlow スイッチ内のフローエントリと不正通信検知システムが検知した攻撃者情報との整合性をとる方法

OpenFlow スイッチ内のフローエントリと不正通信検知システムが検知した攻撃者情報との整合性をとる方法として，2 つの方法を検討している．1 つ目は Master 障害後に Slave が OpenFlow スイッチに一番最近挿入されたフローエントリを問い合わせ，その攻撃者の後に検知した攻撃者から遮断登録する方法である．2 つ目は Master 障害後に Slave が OpenFlow スイッチ内の攻撃者遮断に関するすべてのフローエントリを削除し，保持している攻撃者情報を

再挿入する方法が考えられる．1 つ目の方法のメリットとして，2 つ目の方法と比較すると，再度挿入するフローエントリ数が少ない．デメリットとして，OpenFlow スイッチ内のフローエントリは挿入順に保持されているとは保証されないため，設定時間以外の要素から見分ける必要がある．2 つ目の方法のメリットとして，すべてのフローエントリを再挿入するため，OpenFlow スイッチ内のフローエントリと不正通信検知システムが検知した攻撃者情報との整合性がとれる．デメリットとして，フローエントリの再挿入処理に時間が掛かる．

4. OpenFlow を用いた不正通信制御におけるコントローラ冗長化手法の性能評価

冗長化していない場合，OpenFlow コントローラに障害が起こり OpenFlow スイッチとの接続性が失われると，OpenFlow スイッチの仕様上，OpenFlow スイッチ内のフローエントリが消失する．そのため，OpenFlow ネットワーク全体が停止する危険性がある．また，攻撃者遮断登録も不可能となることが考えられる．

そこで，本章では OpenFlow コントローラにコントローラプロセスのダウンやセキュアチャネルの切断などの障害が起きた場合に本冗長化手法を用いることで，コントローラ優先度が最上位である Slave が Master に切り替わり，OpenFlow ネットワークが停止することなく，正常に攻撃者遮断登録ができているか検証する．動作検証後，Slave が Master へ切り替わる時に登録できなかった攻撃者数と死活管理メッセージ間隔が与えるコントローラ負荷の関係性から適切な送信間隔，コントローラ切り替え時間を考察する．また，冗長化していない場合と本冗長化手法の二通りの結果を比較する．

4.1 実験環境

実験環境として，mininet 2.1.0 を用いて仮想ホスト 3 台，Openvswitch version1.10[12] 1 台，OpenFlow コントローラ 3 台で図 4 の環境を構築した．図 5 に本実験で使用した機器などの性能を示す．OpenFlow コントローラのフレームワークとして Ryu 3.20[7] を用いた．OpenFlow のバージョンは 1.3 である．本実験において，不正通信検知システムの代わりとして，vhost3 が OpenFlow コントローラ群に攻撃者情報を通知する．また，Openvswitch では，スイッチとコントローラが 1 対 1 の場合，コントローラとの接続性が喪失するとフローエントリはすべて消失するが，Multiple Controllers 機能を用いた 1 対多の場合はすべてのコントローラの接続性が喪失しない限り，フローエントリは消失しない．

フローエントリの初期状態として特定の通信を許可するため，あらかじめ図 6 で示すフローエントリを設定する．

図 6 に示すフローエントリは次の意味を持つ．1，2 番目は vhost1 と vhost2 間の通信を許可，3 から 8 番目は，vhost3(不正通信検知システム) と各 OpenFlow コントローラとの通信を許可，9 から 14 番目は各 OpenFlow コントローラ同士の通信を許可，15 番目はフローエントリに該当しないすべての通信を遮断する．評価項目として，vhost1 から vhost2 への ping による疎通確認と OpenFlow スイッチ内のフローエントリから OpenFlow ネットワークの状態を確認し，考察する．

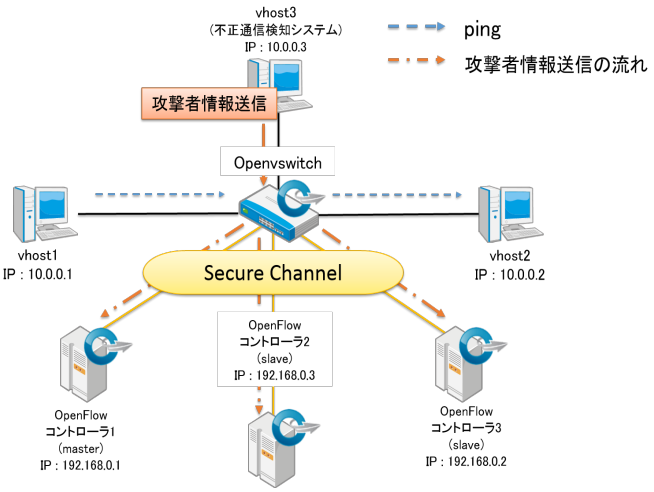

図 4　実験環境

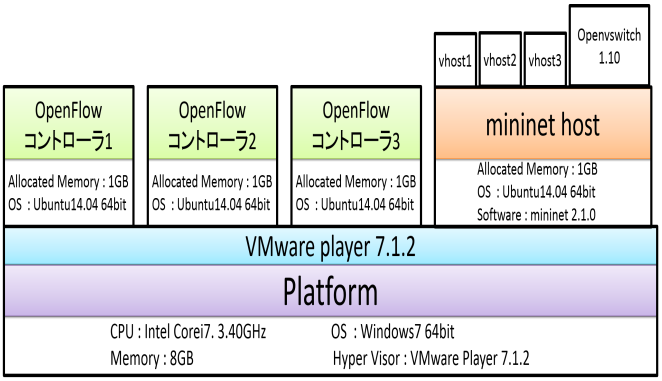

図 5　実験に使用した機器の構成

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
 [1] cookie=0x1, priority=1,ip,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=NORMAL
 [2] cookie=0x2, priority=1,ip,nw_src=10.0.0.2,nw_dst=10.0.0.1 actions=NORMAL
 [3] cookie=0x3, priority=1,ip,nw_src=10.0.0.3,nw_dst=192.168.0.1 actions=NORMAL
 [4] cookie=0x4, priority=1,ip,nw_src=10.0.0.3,nw_dst=192.168.0.2 actions=NORMAL
 [5] cookie=0x5, priority=1,ip,nw_src=10.0.0.3,nw_dst=192.168.0.3 actions=NORMAL
 [6] cookie=0x6, priority=1,ip,nw_src=192.168.0.1,nw_dst=10.0.0.3 actions=NORMAL
 [7] cookie=0x7, priority=1,ip,nw_src=192.168.0.2,nw_dst=10.0.0.3 actions=NORMAL
 [8] cookie=0x8, priority=1,ip,nw_src=192.168.0.3,nw_dst=10.0.0.3 actions=NORMAL
 [9] cookie=0x9, priority=1,ip,nw_src=192.168.0.1, nw_dst=192.168.0.2 actions=NORMAL
 [10] cookie=0xa, priority=1,ip,nw_src=192.168.0.1, nw_dst=192.168.0.3 actions=NORMAL
 [11] cookie=0xb, priority=1,ip,nw_src=192.168.0.2, nw_dst=192.168.0.1 actions=NORMAL
 [12] cookie=0xc, priority=1,ip,nw_src=192.168.0.2, nw_dst=192.168.0.3 actions=NORMAL
 [13] cookie=0xd, priority=1,ip,nw_src=192.168.0.3, nw_dst=192.168.0.1 actions=NORMAL
 [14] cookie=0xe, priority=1,ip,nw_src=192.168.0.3, nw_dst=192.168.0.2 actions=NORMAL
 [15] cookie=0x0, priority=0 actions=drop
```

図 6　フローエントリの初期状態

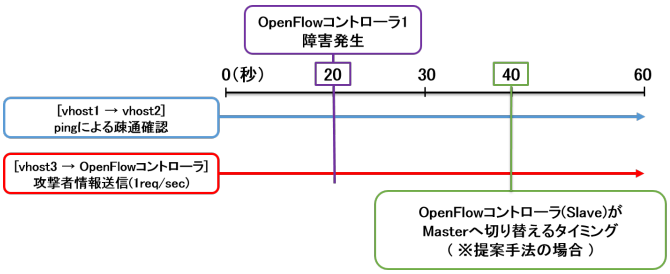

図 7　実験 1：タイムチャート

```
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.177 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.544 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.201 ms
⋮
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=0.137 ms
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=0.137 ms

--- 10.0.0.2 ping statistics ---
60 packets transmitted, 20 received, 67% packet loss, time 61154ms
rtt min/avg/max/mdev = 0.132/0.189/0.544/0.113 ms, pipe 4
```
障害発生

図 8　[実験 1]ping の実行結果：冗長化していない場合

4.2　実験 1：OpenFlow コントローラ (Master) のプロセスダウン時の検証

実験 1 では Master のコントローラプロセスがダウンした場合，Slave が障害を検知し，コントローラ属性を Master に昇格することにより，OpenFlow ネットワークを停止させずに攻撃者遮断登録を引き続き実現できているか検証する．実験手順として，図 7 のタイムチャートに従い，実験する．

- vhost1 から vhost2 へ 60 秒間 ping による疎通確認．
- vhost3 から OpenFlow コントローラ群へ 1 秒間に 1 つの攻撃者情報（IP アドレス）を送信．

4.3　実験 1：実験結果

4.3.1　冗長化していない場合

ping の実行結果を図 8 に，Master 障害前後のフローエントリを図 9，図 10 に示す．図 8 の ping の結果から Master 障害後 (20 秒後) 以降，vhost1 と vhost2 間の疎通性が喪失していることを確認した．また，20 秒前後のフローエントリである図 9 と図 10 より，Master と OpenFlow スイッチとの接続性が失われると，フローエントリがすべて消失していた．これらの実験結果より，冗長化をしていない場合，OpenFlow コントローラに障害が起こり OpenFlow スイッチとの接続性が失われると，フローエントリが消失し，OpenFlow ネットワーク全体の停止を確認した．

4.3.2　本冗長化手法の場合

ping の実行結果を図 11 に，Master 障害前後のフローエントリを図 12，図 13 に示す．図 11 の ping の結果から Master 障害後も vhost1 と vhost2 間の通信ができていた．また，20 秒前後のフローエントリである図 12 と図 13 より，Master と OpenFlow スイッチとの接続性が失われても，フローエントリが保持できていた．さらに，表 4 より，Master 障害から約 20 秒後，再びフローエントリの挿入が開始されていることから Slave であったコントローラが Master へ昇格し，Master として攻撃者を登録していた．これらの結果から，Master のプロセスがダウンして，Slave が障害を検知し，コントローラ属性を Master に昇格することで，OpenFlow ネットワークを停止させずに攻撃者遮断登録を引き続き行えていることを確認した．

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0xf, priority=1000,ip,nw_src=192.168.1.1 actions=drop
 cookie=0x10, priority=1000,ip,nw_src=192.168.1.2 actions=drop
 cookie=0x11, priority=1000,ip,nw_src=192.168.1.3 actions=drop
 ⋮
 cookie=0x20, priority=1000,ip,nw_src=192.168.1.18 actions=drop
 cookie=0x21, priority=1000,ip,nw_src=192.168.1.19 actions=drop
 cookie=0x1, priority=1,ip,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=NORMAL
                                  ↕初期フローエントリー部省略
 cookie=0x0, priority=0 actions=drop
```

図 9　[実験 1] フローエントリ（コントローラ障害前 [19 秒時]）：冗長化していない場合

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
```

図 10　[実験 1] フローエントリ（コントローラ障害後 [21 秒時]）：冗長化なしの場合

```
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.177 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.103 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.101 ms
⋮
64 bytes from 10.0.0.2: icmp_seq=58 ttl=64 time=0.107 ms
64 bytes from 10.0.0.2: icmp_seq=59 ttl=64 time=0.033 ms
64 bytes from 10.0.0.2: icmp_seq=60 ttl=64 time=0.136 ms

--- 10.0.0.2 ping statistics ---
60 packets transmitted, 60 received, 0% packet loss, time 58996ms
rtt min/avg/max/mdev = 0.027/0.127/0.458/0.102 ms
```

図 11　[実験 1]ping の実行結果：本冗長化手法の場合

```
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0xf, priority=1000,ip,nw_src=192.168.1.1 actions=drop
 cookie=0x10, priority=1000,ip,nw_src=192.168.1.2 actions=drop
 cookie=0x11, priority=1000,ip,nw_src=192.168.1.3 actions=drop
 ⋮
 cookie=0x20, priority=1000,ip,nw_src=192.168.1.18 actions=drop
 cookie=0x21, priority=1000,ip,nw_src=192.168.1.19 actions=drop
 cookie=0x1, priority=1,ip,nw_src=10.0.0.1,nw_dst=10.0.0.2 actions=NORMAL
                                  ↕初期フローエントリー部省略
 cookie=0x0, priority=0 actions=drop
```

図 12　[実験 1] フローエントリ（コントローラ障害前 [19 秒時]）：本冗長化手法の場合

表 3　[実験 1]OpenFlow スイッチ内のフローエントリ保持数 (10 秒毎)

時間 (秒)	10	20	30	40	50	60
フローエントリ数 (件)	10	20	20	20	30	40

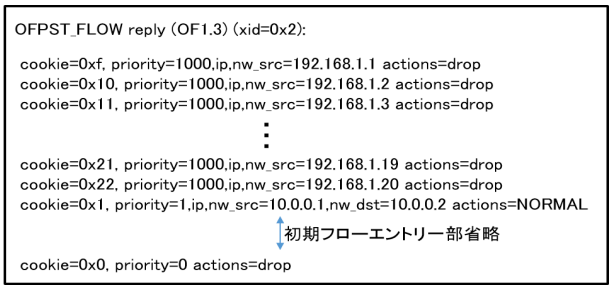

図 13 [実験 1] フローエントリ（コントローラ障害前 [21 秒時]）：本冗長化手法の場合

4.4 実験 2：OpenFlow コントローラ (Master) と OpenFlow スイッチとのセキュアチャネルが切断した場合

実験 2 では図 4 の構成において Master と OpenFlow スイッチとのセキュアチャネルが切断した場合，Slave が障害を検知し，コントローラ属性を Master に昇格することにより，OpenFlow ネットワークを停止させずに攻撃者遮断登録を引き続き実現できているか検証する．実験手順として，実験 1 と同様に図 7 のタイムチャートに従い実験する．実験 2 での障害は Master 側のインタフェースを ifconfig コマンドによりダウンさせることでセキュアチャネルの切断をする．また，本冗長化手法では Master 障害後に Slave が Master へコントローラ属性を昇格させ，Master として攻撃者遮断登録できているか示すため，10 秒毎の OpenFlow スイッチ内のフローエントリ数を計測した．計測時には初期状態のフローエントリは数えない．

4.5 実験 2：実験結果

4.5.1 冗長化していない場合

実験 1 の実験結果と同様，OpenFlow コントローラの障害後は vhost1 から vhost2 への ping が届かなくなり，さらにフローエントリも消失していることを確認した．

4.5.2 本冗長化手法の場合

ping の実行結果を図 14 に，Master 障害前後のフローエントリを図 15，図 16 に示す．図 14 の ping の結果から Master と OpenFlow スイッチ間のセキュアチャネルが切断されても，vhost1 と vhost2 間の通信ができていた．また，20 秒前後のフローエントリである図 15 と図 16 より，Master と OpenFlow スイッチ間のセキュアチャネルが切断されても，フローエントリが保持できていた．さらに，表 5 より，Master の障害から約 20 秒後，再びフローエントリの挿入が開始されていることから Slave であったコントローラが Master へ役割を昇格させ，Master として攻撃者の登録をしていた．これらの結果から，Master と OpenFlow スイッチのセキュアチャネルが切断された場合，コントローラ優先度が最上位である Slave がコントローラ属性を Master に昇格することにより，OpenFlow ネットワークを停止させずに攻撃者遮断登録を引き続き行えていることが確認できた．

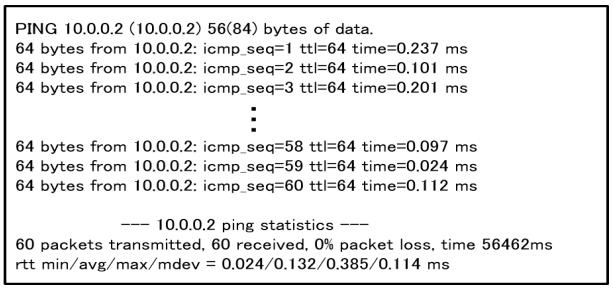

図 14 [実験 2]ping の実行結果：本冗長化手法の場合

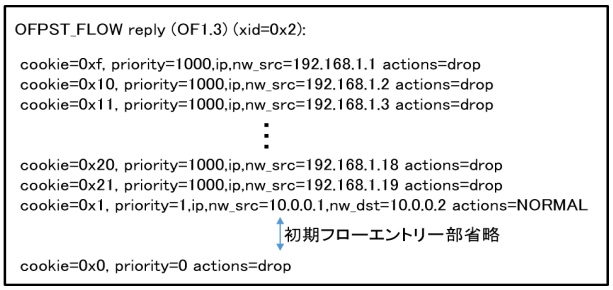

図 15 [実験 2] フローエントリ（コントローラ障害前 [19 秒時]）：本冗長化手法の場合

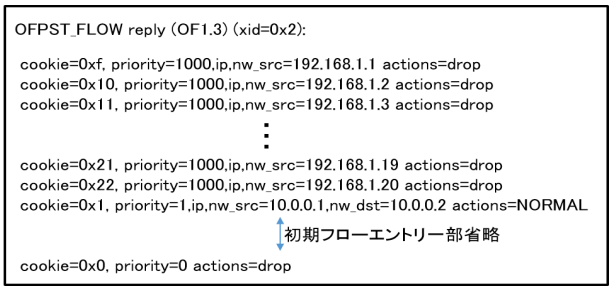

図 16 [実験 2] フローエントリ（コントローラ障害前 [21 秒時]）：本冗長化手法の場合

表 4 [実験 2]OpenFlow スイッチ内のフローエントリ保持数 (10 秒毎)

時間 (秒)	10	20	30	40	50	60
フローエントリ数 (件)	10	20	20	20	30	40

4.6 死活管理メッセージの適切な送信間隔，コントローラ切り替え時間についての考察

Master 障害時に Slave が Master へ切り替わる際に登録漏れをする攻撃者数と死活管理メッセージの送信間隔，死活管理メッセージ送信間隔とコントローラ負荷の関係性から適切な送信間隔，コントローラ切り替え時間を考察する．

本冗長化手法を学内ネットワークで運用するにあたり，Slave が Master へ切り替わる際に登録漏れをする攻撃者数を予測するため，2015 年 1 月 1 日から 2015 年 8 月 31 日の間に不正通信検知システムが検知した攻撃者数（1 秒毎）

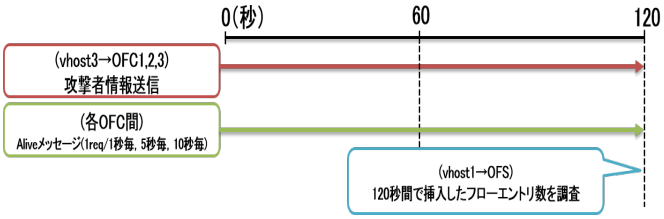

図 17　タイムチャート

を調査した．不正通信検知システムでは 5 分毎に攻撃者件数を記録しているため，5 分間の攻撃者数を 300 秒で割ることで 1 秒毎の攻撃者数とした．結果として，最大は 1 秒に 16 件，最小は 10 秒に 1 件 (0.1 件/秒) であり，平均は 1 秒に 1 件であった．Slave が Master へ切り替わる際に登録漏れをする攻撃者数は，現在のコントローラ切り替え時間 (20 秒) から計算すると，最大で 320 件，平均で 20 件である．コントローラ切り替え時間を調整し小さくすることで再び挿入する攻撃者数は減少すると考えられるが，コントローラ切り替え時間を短くすることで死活確認メッセージを送る回数が増加する．そのため，OpenFlow コントローラに掛かる負荷が大きくなる可能性がある．

そこで死活管理メッセージ送信間隔による，単位時間あたりのフローエントリ数を調査することで死活管理メッセージとコントローラ負荷の関係性を調べた．実験概要として，4 章の性能検証と同様に，図 4 の実験環境，表 5 の機器構成にて実験した．実験手順として，図 17 のタイムチャートに従う．また，実験は死活管理メッセージの送信時間間隔ごとに 10 回試行し，その平均を計算した．死活管理なしの場合の 120 秒間に挿入されたフローエントリ数を基準とし，死活管理ありの場合の挿入フローエントリ数と比較することで OpenFlow コントローラに与える負荷を調査した．図 18 から，本手法の死活管理メッセージ送信間隔で定めている値である 5 秒間隔での挿入フローエントリ数は死活管理なしの場合の挿入フローエントリ数の約 98%を 120 秒間で挿入できていることが確認できた．また，その数秒後残りのフローエントリの挿入は完了している．これらの結果から，死活管理が OpenFlow コントローラに与える影響は小さいと考えられる．

5. おわりに

5.1 まとめ

本論文では，OpenFlow スイッチと OpenFlow コントローラの接続性喪失時に，OpenFlow スイッチ内のフローエントリが消失することで，OpenFlow ネットワーク全体が停止する問題点，攻撃者遮断登録が継続不可能になる問題点を改善するため，OpenFlow コントローラの冗長化手法を提案した．

本冗長化手法の動作検証として，OpenFlow コントローラにコントローラプロセスダウンやセキュアチャネルの喪失などの障害が起きた場合に本冗長化手法を用いることで，Slave が Master へと昇格し，OpenFlow ネットワークが停止することなく，攻撃者遮断登録が正常に行えているか検証した．実験結果より，OpenFlow コントローラのコントローラプロセスダウンやセキュアチャネルの喪失時には，Slave が Master の障害を検知し，コントローラ属性を Master に昇格することにより，OpenFlow ネットワークを停止させずに，継続して攻撃者遮断登録ができていることを確認できた．これらの検証より，OpenFlow を用いた不正通信制御におけるコントローラ冗長化手法は有用性があると考えられる．

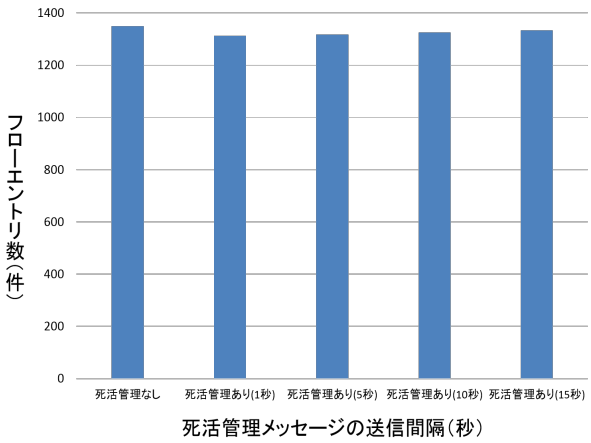

図 18　死活管理のメッセージ送信間隔ごとの単位時間当たりの挿入フローエントリ数

5.2 今後の課題

今後の課題として以下の 2 点に取り組む．

- OpenFlow コントローラ障害後の OpenFlow スイッチ内のフローエントリと不正通信検知システムが検知した攻撃者情報との整合性を取るため，3.3 節の (4) で示した，コントローラ障害発生時から Slave が Master へ切り替わるまでの間に登録漏れをした攻撃者を再度挿入する仕組みが必要となってくる．
- 3 章で示した OpenFlow コントローラ同士がセキュアチャネルとは別のネットワークを介して通信する環境を考慮した冗長化手法が必要となる．上記の環境において，ネットワーク障害などで Master のプロセスが生きている状態でコントローラ群を離脱し，後にコントローラ群へ復帰した場合，Master が二台になり競合してしまう可能性がある．その対処方法として，コントローラ群を離脱した場合にそれを自動で検知し，Master から Slave へ降格させる仕組みを考えなければならない．

参考文献

[1] 警察庁: 情報技術解析平成 26 年報・平成 26 年中のインターネット観測結果等,http://www/npa.go.jp/cyberpolice/ detect/pdf/H26_nenpo.pdf, 2015 年 3 月
[2] 小刀祢知哉，天本大地，小埜勇貴，有馬竜昭，池部実，吉田和幸：scan 攻撃検知システムを用いた被検知ホストの挙動についての調査. 第 65 回電気関係学会九州支部連合大会, pp. 1–1, 2012 年 9 月
[3] 下川大貴，小刀祢知哉，池部実，吉田和幸: OpenFlow を用いた攻撃者遮断システムの提案と評価. 情報処理学会マルチメディア, 分散, 協調とモバイル (DICOMO2014) シンポジウム, pp.197-204, 2014 年 7 月
[4] 下川大貴，小刀祢知哉，池部実，吉田和幸: OpenFlow を用いた不正通信制御システムの評価. インターネットと運用技術シンポジウム 2014[WIP], pp.31-31, 2014 年 12 月
[5] Open Networking Foundation，http://www.opennetworking.org/
[6] OpenFlow Switch Specification version1.2，https://www.open networking.org/images/stories/downloads/sdn-resources/onf- specifications/openflow/openflow-spec-v1.2.pdf
[7] Ryu : component-based software defined networking framework，http://osrg.github.io/ryu/
[8] Floodlight : component-based software defined networking framework，http://www.projectfloodlight.org/floodlight/
[9] OpenDaylight : component-based software defined networking framework，https://www.opendaylight.org/
[10] 黒木圭介, 林通秋: OpenFlow Controller における冗長化についての考察. 電子情報通信学会通信ソサイエティ大会, pp. 1150-115, 2012 年 9 月
[11] 黒木圭介, 林通秋：OpenFlow Controller における 1:1 冗長化手法の検討. 電子情報通信学会総合大会, pp. 225–225, 2013 年 3 月
[12] Open vSwitch: http://openvsw-itch.org/