OpenFlow ネットワークにおける Packet-In メッセージの特性に基づくポートスキャン検出手法の設計と実装

小野 大地[1]　和泉 諭[2]　阿部 亨[1,3]　菅沼 拓夫[1,3]

概要：
不正アクセスやマルウェアによるサイバー攻撃の脅威が増大しており，それらへの対策は喫緊の課題となっている．一方，ネットワークをソフトウェアによって柔軟に制御・管理する技術として Software Defined Network (SDN) が注目されており，具体的な実装として OpenFlow が広く用いられている．OpenFlow はスイッチからトラフィックに関する統計情報を収集することが可能であることに加え，スイッチレベルで通信の遮断が可能であることから，ネットワークのサイバーセキュリティ対策に有用であることが過去の研究で示されている．本研究ではサイバー攻撃の準備段階として行われるポートスキャンの検出に注目し，OpenFlow 環境下におけるポートスキャン検出に関して，オーバーヘッドの増加や検出遅延に関する課題を解決するための手法を提案する．具体的には，既存の周期的にポートスキャン検出処理を行う手法に対して，Packet-In メッセージの特徴を考慮したイベントドリブンな検出手法の設計・実装を行い，より迅速かつ低いオーバーヘッドでの検出の実現を目指す．また，実験により，実際の正常なトラフィックにおける誤検出の頻度に関する評価や，既存手法とのパフォーマンスの比較，検出遅延の測定を行い，提案手法の有用性を示す．

A Design and Implementation of Port Scan Detection Method Based on the Characteristics of Packet-In Messages in OpenFlow Networks

1. はじめに

不正アクセスやマルウェアによるサイバー攻撃の脅威が年々増大しており，機器やネットワークに対するサイバー攻撃に関連するトラフィックもまた増加傾向にある．そのため，悪意のあるネットワーク利用を迅速に検出し，不正なホストをネットワークから遮断することがセキュリティ対策をする上で増々重要となっている．

一般的に，マルウェアに感染したホストは感染をさらに拡大させるために，他のホストが使用しているポート番号やオペレーティングシステム，アプリケーションなどをスキャンし，脆弱性を見つけ出そうとする．このような行為を「ポートスキャン」と呼ぶ．このポートスキャンを迅速に検出し，ホストをネットワークから遮断することで，マルウェアに感染したホストや悪意のあるユーザによる被害の拡大を最小限に抑えることが可能である．

ポートスキャンを検出・防御するために様々な手法が提案されてきているが，いまだ課題が存在する．ネットワークの結節点にファイアウォールを設置してトラフィックを監視する手法ではイントラネット内で発生するポートスキャンへの対応が不可能である．また，Intrusion Detection System (IDS) や Intrusion Prevention System (IPS) により，ネットワークを流れる全てのトラフィックを監視する手法では，大規模化するトポロジやトラフィック量に対して対応が難しいという問題点がある．そこで，Software Defined Network を用いてこれらの問題を解決する試みが行われた [1].

SDN は，ネットワークをソフトウェアによって柔軟に制御・管理する技術であり，従来のネットワークに変わる

[1] 東北大学大学院情報科学研究科
Graduate School of Information Sciences, Tohoku University
[2] 仙台高等専門学校総合工学科
National Institute of Technology, Sendai College
[3] 東北大学サイバーサイエンスセンター
Cyberscience Center, Tohoku University

新たな仕組みとして注目されている．また，OpenFlow[2] による実現が広く普及している．加えて，近年ではデータセンターだけでなく，ビルや建物内のオフィスネットワーク内の構成を集中管理したり柔軟に構成を変更したりといった用途で SDN を用いる動きがある [3], [4]．SDN 環境では，トラフィックのルーティングが従来とは異なる形式で行われたり，スイッチでパケットの書き換えが行われたりする可能性があり，従来型の IDS/IPS の適用が難しい場合がある．したがって，SDN 環境に最適な新たなセキュリティ対策を考える必要がある．

OpenFlow は，OpenFlow 対応スイッチからトラフィックに関する統計情報を収集することが可能であり，また，スイッチ単位での通信の遮断が可能であることから，ネットワークのサイバーセキュリティ対策に有用であることが示されている [1], [5], [6] が，処理負荷によるオーバーヘッドや迅速な検出に関しては課題が存在する．

本研究では，SDN 環境下かつパーソナルコンピュータ等によって構成されるオフィスネットワークを想定環境とした，ポートスキャン検出の高度化を目的とする．具体的には，OpenFlow スイッチからコントローラに対して送信される Packet-In メッセージの特徴を考慮した，ポートスキャンの検出手法を提案する．

本手法は，既存研究 [1] に関連して，トラフィックの統計情報だけでなく，OpenFlow スイッチからコントローラに対して送信される Packet-In メッセージの流量をモニタリングし，その流量の変化をもとにポートスキャンを検出する．これにより，常時，全てのスイッチの統計情報を収集して検出する従来の手法と比較して，より迅速かつ低いオーバーヘッドでポートスキャンの検出を行うことができる．さらに，研究室で収集した実際のトラフィックデータを用いた分析やシミュレーション実験により，提案手法の評価を行った．実験結果より，提案手法では既存手法に対してより低いオーバーヘッドでポートスキャンを検出できることを確認した．

2. 関連研究

2.1 SDN と OpenFlow

SDN のアーキテクチャの概念を図 1 に示す．SDN は一般に，アプリケーションレイヤ，コントロールレイヤ，インフラストラクチャレイヤにより構成されており，コントロールレイヤとインフラストラクチャレイヤ間のインターフェースとして，OpenFlow[2] が広く用いられている．

OpenFlow は，コントローラとスイッチ間で OpenFlow メッセージをやり取りすることでネットワークを制御する．OpenFlow メッセージには，スイッチからコントローラにパケットの処理方法を問い合わせる Packet-In メッセージや，コントローラからスイッチに経路情報を書き込む Flow-Mod メッセージ，コントローラからスイッチへトラフィックの統計情報の送信をリクエストする Stats-Request メッセージなどがある．

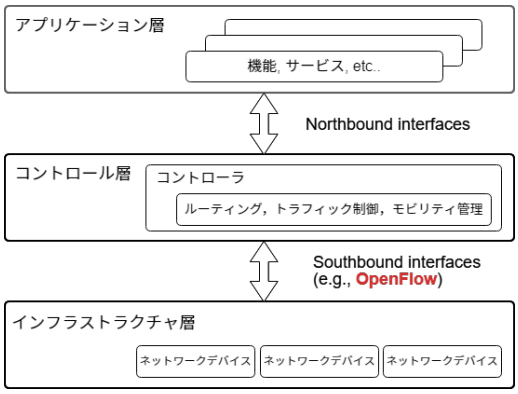

図 1 SDN アーキテクチャ

OpenFlow では，コントローラはスイッチに対して，ヘッダフィールド (マッチ条件) と対応する処理方法 (アクション) が記述されたフローエントリを書き込む．スイッチはパケットを受信すると，パケットがどのフローエントリにマッチするかを探索し，一致するものが存在した場合には，該当するフローエントリに記述されたアクション（転送や遮断など）を実行する．一致するフローエントリが存在しない場合，当該するパケットを Packet-In メッセージによってカプセル化し，コントローラに送信することで処理方法の問い合わせを行う．また，OpenFlow スイッチはフローエントリ毎に統計情報を記録しており，受信パケット数 (カウンタ) や受信バイト数，フローエントリが追加されてからの経過時間等の情報をコントローラで収集することが可能である．

OpenFlow スイッチが管理するフローエントリとカウンタ情報の例を表 1 に示す．

2.2 ポートスキャン

通常，ネットワーク中のシステムに対する攻撃や侵入の前に，攻撃対象の脆弱性を調査するために攻撃者によるスキャンが行われる [7]．特に，攻撃対象のシステムが使用しているポートに関する情報を得るために行われるポートスキャンは主要な手法であり，ポートスキャンを行っている不審なホストを迅速に検出して，ネットワークから遮断することは，さらなる攻撃やマルウェアの感染拡大の防止のために重要となる．

ポートスキャンでは，ネットワーク上のホストのポートに対して特定のデータを送り，その応答を分析することで，ホストで使用されているポート番号や OS, アプリケーションに関する情報を収集する．基本的なポートスキャンの方法としては TCP スキャンが挙げられる．これは対象のポートに TCP の 3 ウェイハンドシェイクを試みることでポート使用の有無を確認する手法である．例えば正常な通信であっても TCP 通信時には 3 ウェイハンドシェイクは

表 1　フローエントリの例

Header fields (Match)						Action	Counter
src Eth	dst Eth	src IP	dst IP	src Port	dst Port		
00:00:00:00:00:01	00:00:00:00:00:02	10.0.0.1	10.0.0.2	10000	443	Forward: 2	100
00:00:00:00:00:02	00:00:00:00:00:01	10.0.0.2	10.0.0.1	443	10000	Forward: 1	50
00:00:00:00:00:02	00:00:00:00:00:01	10.0.0.2	10.0.0.1	22	22	Drop	10
00:00:00:00:00:03	*	*	*	*	*	Send Packet-In	10

行われるため，1 つ 1 つのパケット単位ではポートスキャンと正常な通信の判別は困難である．したがって，ポートスキャンの判別には複数のパケットから特徴を見つけ出す必要があり，同一ホストから異なるホストへの単位時間あたりのポートアクセス数や，1 ポートあたりのトラフィック量などを用いて検出を行う [8], [9]．

2.3 OpenFlow 環境におけるポートスキャン検出に関する研究

OpenFlow 環境におけるポートスキャン検出に関する研究が行われている [1]．この研究では，各 OpenFlow スイッチの持つフローエントリの統計情報を数秒間隔で周期的に収集し，それらを分析することでポートスキャンが発生したかどうかを判定している．具体的には各フローエントリのカウンタ情報と，その頻度をもとにポートスキャンの有無を判定する．前述したように，通常，ポートスキャンによる 1 ポートあたりのトラフィック量は非常に少ない．加えて，このような通信を複数の宛先ポートに対して試みる．すなわち，すべての統計情報の中から，少ない数のパケットを複数のポートに対して送信しているホストが存在しているかどうかを検出するというのが，この手法の基本的なアイディアとなっている．

この手法は数秒間隔で周期的に検出処理を実行しているため，スローポートスキャンにも対応が可能な点が利点と言える．しかし，一般に低い頻度で発生するポートスキャンに対して，何度もポートスキャンの検出処理を行うというのは効率的とは言えない欠点もある．

また，全てのスイッチから定期的に統計情報を収集しているため，ネットワークを構成する OpenFlow スイッチ数の増加に応じて，ポートスキャンの検出処理を行うコントローラの負荷が大きくなる．したがって，この手法の欠点を補う形で，ポートスキャンの検出処理の実行頻度を抑える手法が必要と言える．

2.4 OpenFlow 環境における DDoS 攻撃検出に関する研究

OpenFlow 環境で DDoS 攻撃を検出する手法が提案されている [10], [11]．これらの研究は周期的に統計情報を収集して分析する手法を採っているが，前述したように定期的な収集には様々な課題がある．そこで，スイッチの統計情報だけでなく，Packet-In メッセージの特徴に基づく DDoS 攻撃検出手法が提案されてきた [5], [12]．これらの手法では，一般に DDoS 攻撃は送信元の IP アドレスを偽装して大量のパケットを送信するという点に着目している．OpenFlow スイッチはフローエントリにマッチしない未知のトラフィックに対しては Packet-In メッセージを発生させるため，IP Spoofing を用いた DDoS 攻撃によるトラフィックは，局所的に大量の Packet-In メッセージを発生させる．したがって，Packet-In メッセージの急増加やエントロピーの増加を検出することでより効率的に DDoS 攻撃を検出することが可能であることが示された．しかし，ポートスキャン検出における適用可能性や，実際の正常なトラフィックにおける検出処理の誤検出の可能性については十分な検証がなされていない．

本研究では，このアプローチを基本として，Packet-In メッセージの特徴を活用し，効率的なポートスキャンの検出を検討する．

3. Packet-In メッセージの特性に基づくポートスキャン検出手法の提案

3.1 概要

本研究では，2.2 節で述べた課題に対する新たなポートスキャンの検出手法を提案する．

ポートスキャンは一般に，短時間で複数のポートに対する多くのパケットを発生させる．OpenFlow 環境において，スイッチが初めて受信する種類のパケットに対して Packet-In メッセージを発生させることが可能であるため，ポートスキャンが行われた場合，Packet-In メッセージの局所的な増加が発生する．そこで，本研究ではこの特徴に着目し，Packet-In メッセージの異常増加が見られた場合を起点に，ポートスキャンの検出処理に移行する手法を提案する．

提案手法のフローチャートを図 2 に示す．提案手法では，各スイッチからコントローラに対して送信される Packet-In メッセージの流量を，各ホストが持つ Ethernet アドレス毎にモニタリングし，流量の異常増加が見られた場合に，当該ホストが接続されているスイッチに対して統計情報の送信をリクエストする．その後，収集された統計情報をもとに，既存研究 [1] で提案された手法をベースにポートスキャンの有無を判定する．この手法は，ポートスキャンに起因するイベントの発生を確認してから検出処理に移行するため，ポートスキャン検出処理を周期的に行う方法と比

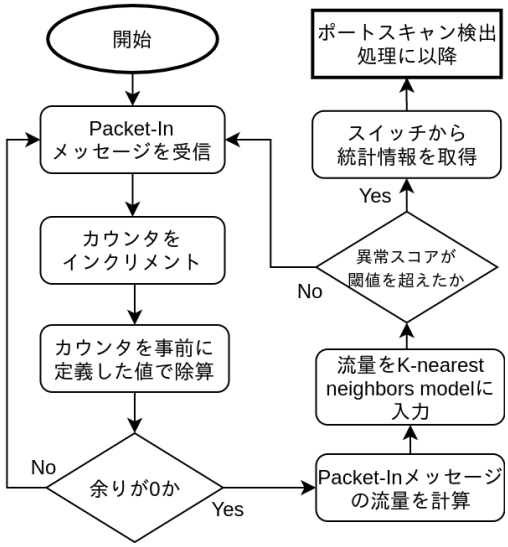

図 2 提案手法のフローチャート

較して検出処理の実行頻度が少なくなり，オーバーヘッドの低減が実現できる．また，特定のスイッチからのみ統計データを収集するので，スイッチ数が多い場合でも迅速な検出処理を行うことができる．

以下では，提案手法の具体的な内容について説明をする．

3.2 Packet-In メッセージのモニタリング

スイッチが未知のパケットを受信した場合，当該パケットは Packet-In メッセージとしてカプセル化され，コントローラに送信される．したがって，Packet-In メッセージからは Packet-In メッセージの発生源となったホストの Ethernet アドレスを得ることが出来る．そこで，提案手法では Ethernet アドレス毎に Packet-In メッセージの到達時刻を記録する．これによって，Packet-In メッセージを大量に発生させているホストの特定が容易になると共に，後に統計情報リクエストする際に対象のスイッチを限定することが可能となる．また，複数のホストによって同時に Packet-In メッセージが発生した場合においても，本手法では流量はホスト毎に測定されるため，誤ってポートスキャン検出処理が発火してしまう可能性を減らすことが可能である．

Packet-In メッセージの流量は単位時間当たりの Packet-In メッセージの数で表され，同一のホストから異なる宛先ポート番号に対する Packet-In メッセージを m 個受信する毎に式 (1) で求められ，記録される．$\Delta Time$ は，現在の時刻と，前回に流量を計算した時刻の差を表す．

提案手法では，単一のホストの複数のポート番号から，単一のホストの一つのポート番号に対する通信で発生する Packet-In メッセージについては，流量の測定対象にはしていない．なぜなら，ポートスキャンにおいて，ポート番号に関する多対一通信が発生することは無く，OS による送信元ポート番号の動的割り当てによる Packet-In メッセージの増加が，誤検出の増加をもたらしてしまうためである．

$$FlowRate : R_p = \frac{m}{\Delta Time} \tag{1}$$

3.3 Packet-In メッセージの異常増加検出

本手法では，記録された Packet-In メッセージの流量をもとに，k 近傍法を用いて異常増加の検出を行う．単純に流量に対する閾値を設定する手法と比較して，k 近傍法などの変化点検出アルゴリズムを用いることで，データの波形変化に対する異常を検出することが可能となる．例えば，ポートスキャンが発生した場合，Packet-In メッセージの流量はしばらくの間は高い値を取り続けると想定される，仮に流量に対して閾値を低く設定した場合は何度も異常が検出されることになる．しかし，変化点検出アルゴリズムを用いた場合，理論的には一度のポートスキャンの発生に対して一度だけポートスキャン検出処理が発火するトリガーを作ることが可能となる．

具体的な処理手順を以下に示す．

Step 1: 流量のデータ列に対してウィンドウ W を設定する

Step 2: 新しいデータが得られた場合，新しいデータと W 内の全てのデータとの差を求める

Step 3: Step2 で計算した差について，最も小さい k 個のデータを選び，その平均値を新しいデータの異常スコアとする

Step 4: 異常スコアが閾値を超えていた場合は異常増加があったと判定する

3.4 統計情報に基づくポートスキャン検出処理

あるホストからの Packet-In メッセージの異常増加が検出された場合，提案システムは Packet-In メッセージの異常増加を発生させているホストの Ethernet アドレスを元に，ホストが接続されているスイッチを特定する．その後に，コントローラからスイッチに対して統計情報のリクエストが行われ，受信した統計情報をもとにポートスキャンの検出処理が行われる．

ポートスキャンの検出アルゴリズムは，既存手法 [1] をベースにしている．既存手法では，ポートスキャンのストリームは通常 3 パケット以内に収まることに着目している．TCP に関する各フローエントリについて，マッチしたパケット数が 5 以下の数をカウントし，頻繁にスキャンされるポート番号とスキャンされないポート番号に対して重みをつけてスコアを算出する．スコアが閾値を超えた場合にはポートスキャンが発生したと判定する．

3.5 ホストの遮断

あるホストがポートスキャンを行っていると判定された場合，コントローラからそのホストが接続されているス

イッチに対して，そのホストの Ethernet アドレスを送信元にアドレスとして持つパケットをドロップするフローエントリを追加する．これ以降，ホストからスイッチの外へのあらゆる通信がそのフローエントリによって破棄されるため，イントラネット内であってもその他のホストへ攻撃やスキャンを行うことを防ぐことができる．

4. 実験

4.1 概要

Packet-In メッセージの特徴を用いたポートスキャン検出機能をモジュールとして実装したコントローラを用意し，提案手法の有効性を以下の 3 つの実験により評価した．

- 実験 1: ポートスキャン発生時の Packet-In メッセージ流量の検証
- 実験 2: 提案手法のパフォーマンス評価
- 実験 3: 提案手法の検出遅延評価

実験 1 では，OpenFlow 環境においてポートスキャン発生時の Packet-In メッセージの流量の大きさを調べた．さらに，研究室のネットワーク環境で収集した平日 5 日分の実トラフィックを用いて分析を行い，Packet-In メッセージが発生した場合の流量の分布を測定した．次に，実験 2 において，提案手法と，定期的にポートスキャン検出処理を行う既存手法（ポーリング型手法）におけるパフォーマンスを比較した．最後に，実験 3 として，提案手法において，ポートスキャンが発生してから検出が完了するまでの遅延を測定した．

4.2 実験環境

実験を行うために Mininet[13] を用いて仮想的な OpenFlow ネットワークを構築した．Mininet はオープンソースのネットワークエミュレータであり，1 つの Linux カーネル上で複数のホストやスイッチをエミュレートし，仮想的なネットワークを構築することができるまた，Mininet 上で立ち上げたホストからは任意のプログラムを実行することが可能である．Mininet 上で用意するスイッチとして OpenFlow に対応した Open vSwitch[14] を用いることで，実際の環境のようなデータプレーンを構築した．コントローラは Ryu[15] を用いて，OpenFlow version 1.3 環境下で動作するコントローラを作成した．

実験のために構築した OpenFlow ネットワークを図 3 に示す．OpenFlow ネットワークは 1 台のコントローラと 15 台のスイッチと 45 台のホストから構成されている．各スイッチは直列に接続され，それぞれ 3 台のホストが接続されている．また，全てのスイッチは制御用のリンクでコントローラと接続され，OpenFlow メッセージの送受信はそのリンク上で行われる．

実験に用いたマシンの性能は，Intel Core i7-8700K @ 3.70 GHz, 16GB RAM (2133 MHz) である．このマシン上

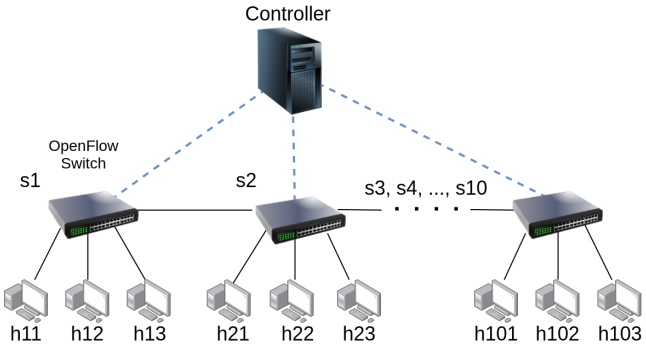

図 3 実験に用いたネットワークのトポロジ

では，Mininet と Virtualbox[16] で作成した仮想マシンが動作しており，仮想マシンには 4 GB の RAM と 2 コアの CPU が割り当てられ，OS として Ubuntu 18.04 がインストールされている．作成したポートスキャン検出機能を持つ OpenFlow コントローラをこの仮想マシン上で動作させ，仮想マシンのパフォーマンスを測定することで評価を行った．

トラフィックのルーティングに関して，各スイッチがそれぞれラーニングスイッチとして動作するようにコントローラを設計した．スイッチはホストからフローエントリに存在しないパケットを受信した時，コントローラに対して Packet-In メッセージを送信する．コントローラは Packet-In メッセージから得られる宛先 Ethernet アドレスを元に，当該スイッチに対して経路情報に関するフローエントリを追加する．

4.3 実験 1: ポートスキャン発生時の Packet-In メッセージの流量の検証

上記の環境で 5 回のポートスキャンを行い，Packet-In メッセージの流量を計測した．実験によって得られた流量の時系列データの例を図 4 に示す．ポートスキャン発生時の Packet-In メッセージの流量のピークは平均して 17,640 Packet-In per second (pps) であった．また図より，ポートスキャン発生時の Packet-In メッセージの流量は極めて短い時間にピークを迎え，その後は 1,200pps 前後の比較的低い数値で推移していることが分かる．

加えて，正常なトラフィック時における Packet-In メッセージの流量の検証を行った．ここでは研究室の学生居室で発生した平日 5 日間のトラフィックを用いた分析を行った．研究室の学生居室では，1 台のルーター配下で 20 台程度の PC が調査や開発，実験等で使用されている．各 PC にはセキュリティソフトの導入を徹底しており，ウィルス等による不正な操作があった可能性は低いと考えられることから本実験では正常なトラフィックとして扱うこととした．収集したトラフィック情報を解析し，OpenFlow ネットワーク内に流れた場合の Packet-In メッセージの流量をシミュレーションにより算出した．

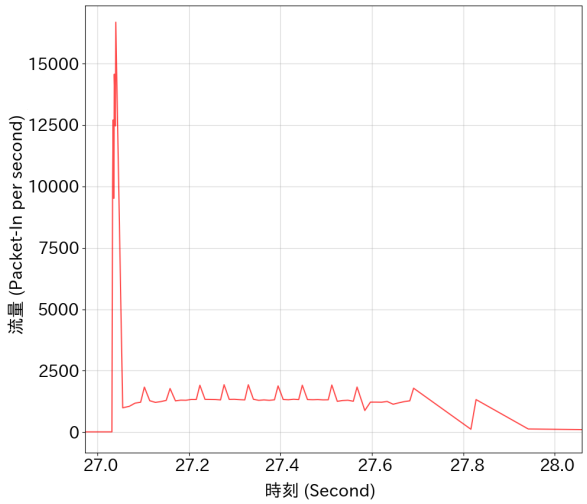

図 4　ポートスキャン発生時の Packet-In メッセージの流量

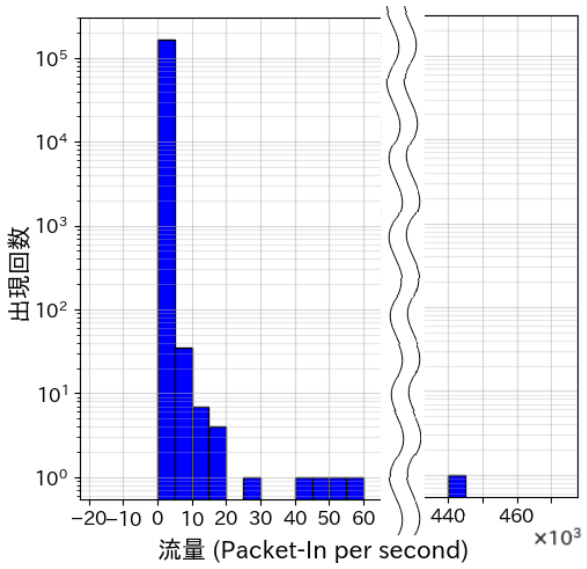

図 5　実際のトラフィックにおける Packet-In メッセージの流量の分布

実験の結果として，実際の正常なトラフィックにおける Packet-In メッセージの流量の分布を図 5 に示す．ここで，横軸は記録された流量の値を表し，縦軸はそれらの流量が記録された回数を表す．縦軸は対数スケールとなっている．記録された流量のうち，97%は 100 pps 以下であり，流量の中央値は 1.83 pps であった．また，流量が 10,000 pps を超えた回数は，5 日間で 17 回であった．

これらの結果から，Packet-In メッセージの流量は，ポートスキャン発生時と平時とでは大きく隔たりがあり，正常トラフィックに対する誤検出の頻度は高くはないことが分かった．したがって，Packet-In メッセージの流量をポートスキャン検出のトリガーとして用いるアプローチは実環境においても有用であると言える．

4.4　実験 2: 提案手法のパフォーマンス評価

実験 2 では，コントローラの各時刻におけるパフォーマンスをモニタリングし，CPU 使用率と 1 秒間あたりの送受信パケット数の関係を調査した．実験にあたって，各スイッチにある程度の統計情報が蓄積されていることを想定するために，各スイッチには事前に 1,000 個のフローエントリを追加した．各フローエントリは宛先ホストと送信元ポート番号と宛先ポート番号がランダムに決定されたものであり，それぞれ正常にルーティングが行われる経路を示す．

表 2　提案手法で用いたパラメータ

パラメータ	値	説明
m	15	流量の計算処理を行う間隔
W	10	k 近傍法のウィンドウサイズ
k	3	k 近傍法で用いる近傍数
T	8000	異常スコアに対する閾値

また，4.2 節の実験から，Packet-In メッセージの流量の中央値は 1.83 pps であることがわかったことから，各ホストで Packet-In メッセージを 2.0 pps の流量で発生させるスクリプトを実行し続け，コントローラに対して実際のネットワークと同様の負荷を与えた．ポーリング型手法について，検出処理の実行間隔は 10 秒と設定した．また，提案手法で用いた各種パラメータを表 2 に示す．

実験は，ポーリング型手法と提案手法の両方で 30 分間ずつ行い，それぞれ指定の時刻にポートスキャンを発生させた．ポーリング型手法は，02:15:00 時点からパフォーマンスの測定を開始し，02:20:00 と 02:30:00 と 02:40:00 の時点でポートスキャンを発生させた．また，提案手法では，03:40:00 時点からパフォーマンスの測定を開始し，03:45:00 と 03:55:00 と 04:05:00 の時点でポートスキャンを発生させた．

実験の結果として，ポーリング型手法の CPU 使用率に関するグラフを図 6 に，提案手法の CPU 使用率に関するグラフを図 7 にそれぞれ示す．グラフから，ポーリング型手法ではコントローラの CPU 使用率が頻繁に急上昇しているのに対して，提案手法ではポートスキャンの発生時刻では CPU 使用率が急上昇しているが，それ以外の時刻では基本的に 40%以下となっている．

次に，ポーリング型手法の 1 秒間あたりの送受信パケット数に関するグラフを図 8 に，提案手法の 1 秒間あたりの送受信パケット数に関するグラフを図 9 にそれぞれ示す．グラフから，CPU 使用率と同様に，ポーリング型手法では頻繁に 1 秒間あたりの送受信パケット数が上昇しているのに対して，提案手法ではポートスキャン発生時のみ増加していることが確認できる．

コントローラにおける CPU 使用率や 1 秒間あたりの送受信パケット数の急増加は，OpenFlow ネットワークにおけるスイッチング処理に影響を及ぼす可能性がある．実際，ポーリング型手法の実験では，これらの急増加時にコントローラからスイッチに対する Flow-Mod メッセージによる

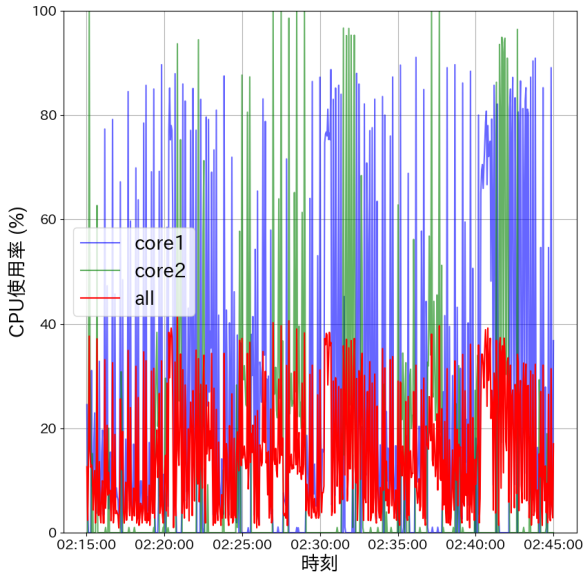

図 6 ポーリング型手法における CPU 使用率

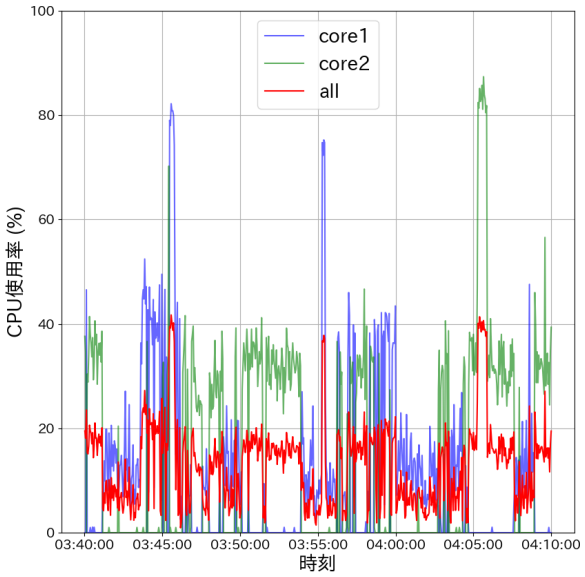

図 7 提案手法における CPU 使用率

フローエントリの追加に遅延が生じ，フローエントリが追加されるまでの間に発生した全てのパケットが Packet-In メッセージとしてコントローラに送られてくるという現象が見られた．提案手法はポーリング型手法に対して，コントローラにおける CPU 使用率や 1 秒間あたりの送受信パケット数のスパイクを減らすことができていることがグラフから確認できた．従って，提案手法により，ネットワークのオーバーヘッドの低減を確認できた．

考察として，提案手法では Packet-In メッセージの流量の異常増加を見ているため，非常に低レートで行われるポートスキャンに対しては有効ではない．その点，既存手法 [1] ではこのようなポートスキャンにも対応が可能であるが，ポーリング間隔を小さく取るとオーバーヘッドが大きいことは今回の実験で示した．したがって，既存手法 [1] のポーリング間隔を大きく取ったものとの併用をすることで，より実用的なシステムへと発展させることが出来ると考える．

4.5 実験 3: 提案手法の検出遅延評価

実験 3 として，提案手法のポートスキャンが発生してから検出されるまでの遅延について評価を行った．実験では，実験 2 と同様に，各スイッチに 1000 件のフローエントリが追加され，各ホストから継続的に Packet-In メッセージが発生している状態で，間隔をあけて 5 回のポートスキャンを発生させた．ポートスキャンは，図 3 のホストについて，h11 から h31 へ，h41 から h101 へ，h111 から h151 へ，h11 から h101 へ，h141 から h151 へと行った．実験の結果，ポートスキャンに起因するパケットが Packet-In メッセージとしてコントローラに届いてから検出が完了するまでの遅延は平均で 6.18 秒程度であることが確認できた．実験 24.3 で示したとおり，ポーリング型手法は提案手法と比較してオーバーヘッドが大きく，ポーリング型手法のオーバーヘッドを提案手法と同等にするためには，ポートスキャン検出処理の実行間隔を実験で用いた 10 秒よりも長く設定する必要がある．その点，提案手法ではより低いオーバーヘッドでより低い検出遅延を実現できていることが分かる．

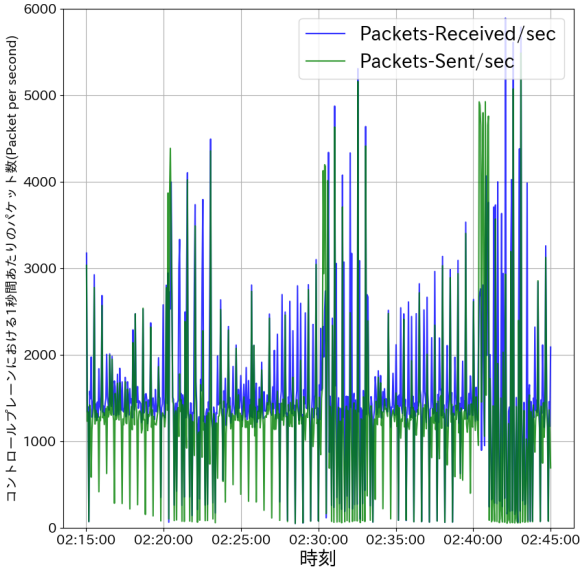

図 8 ポーリング型手法における 1 秒間あたりの送受信パケット数

h11 から h31 に対するポートスキャン時の各処理が行われた時刻を以下に示す．

22:57.95: コントローラがポートスキャンに起因する Packet-In メッセージを受信する
22:59.06: Packet-In メッセージの流量の異常スコアが閾値を超える
22:59.09: スイッチに対して統計情報をリクエスト
23:03.77: スイッチから統計情報を受信
23:03.77: ポートスキャンが検出される

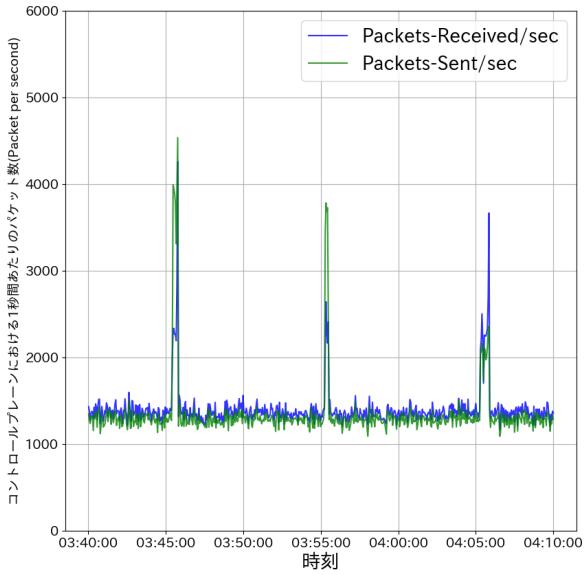

図 9 提案手法における 1 秒間あたりの送受信パケット数

このことから，提案手法の検出遅延の殆どは，スイッチに統計情報をリクエストしてから返ってくるまでの遅延が支配的であることが分かる．

以上，実験結果から提案手法により検出遅延を抑えることができた．

5. おわりに

本研究では，Packet-In メッセージとトラフィックの統計情報に基づくポートスキャン検出システムの設計と実装を行った．さらに提案手法の有効性を検証するための実験を行った．実験の結果，提案手法は実環境においても有用であると共に，既存のポーリング型手法と比較して，CPU 使用率と 1 秒間あたりの送受信パケット数の急上昇を抑えることができ，オーバーヘッドの小さいシステムであることを示した．また，提案手法におけるポートスキャンの発生から検出までの遅延は平均 6.18 秒であった．

現状では Packet-In メッセージの流量の異常増加のみを起点にポートスキャンの検出処理を行っているため，送信レートを落として非常にゆっくりとポートスキャンを行う Slow Scan に対しては効果的ではない．したがって，既存のポーリング型手法や IDS/IPS との併用を考慮し，Slow Scan も検出が可能な手法を検討する．また，OpenFlow メッセージの処理方法や効率は OpenFlow コントローラの実装によって変わるため，今回の実装で用いた Ryu 以外のフレームワークを用いて開発した場合の評価を行う．さらに，より効率的な Packet-In メッセージの異常増加検出手法の考案や検出遅延の低減を目指していく．

参考文献

[1] Neu, C. V., Tatsch, C. G., Lunardi, R. C., Michelin, R. A., Orozco, A. M. S. and Zorzo, A. F.: Lightweight IPS for port scan in OpenFlow SDN networks, *2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6 (2018).

[2] OPEN NETWORKING FOUNDATION: OpenFlow Switch Specification, (online), available from ⟨https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf⟩ (2012).

[3] UNIADEX,Ltd.: オフィスネットワーク向けに高品質で安全なネットワーク環境を安価に提供開始，（オンライン），入手先 ⟨https://www.uniadex.co.jp/news/2018/20181107$_b$cf $-$ $for-campus-core.html$⟩ (2018).

[4] NEC Corporation: 診療科の拡充、機器の追加、レイアウト変更にも即応「変化に強いネットワーク」が病院の成長を支える，（オンライン），入手先 ⟨https://jpn.nec.com/case/okamoto-hp/index.html⟩.

[5] Cui, Y., Yan, L., Li, S., Xing, H., Pan, W., Zhu, J. and Zheng, X.: SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks, *Journal of Network and Computer Applications*, Vol. 68, pp. 65 – 79 (online), DOI: https://doi.org/10.1016/j.jnca.2016.04.005 (2016).

[6] Shimizu, T., Kitagawa, N. and Yamai, N.: Implementation of a Mirroring System for Initial Sequence of Flow Using OpenFlow, *Internet and Operation Technology Symposium 2016*, pp. 83–90.

[7] Bou-Harb, E., Debbabi, M. and Assi, C.: Cyber Scanning: A Comprehensive Survey, *IEEE Communications Surveys Tutorials*, Vol. 16, No. 3, pp. 1496–1519 (2014).

[8] Gadge, J. and Patil, A. A.: Port scan detection, *2008 16th IEEE International Conference on Networks*, pp. 1–6 (2008).

[9] Lee, C. B., Roedel, C. and Silenok, E.: Detection and Characterization of Port Scan Attacks (2003).

[10] Braga, R., Mota, E. and Passito, A.: Lightweight DDoS flooding attack detection using NOX/OpenFlow, *IEEE Local Computer Network Conference*, pp. 408–415 (2010).

[11] Cui, Y., Yan, L., Li, S., Xing, H., Pan, W., Zhu, J. and Zheng, X.: SD-Anti-DDoS: Fast and efficient DDoS defense in software-defined networks, *Journal of Network and Computer Applications*, Vol. 68, pp. 65 – 79 (online), DOI: https://doi.org/10.1016/j.jnca.2016.04.005 (2016).

[12] You, X., Feng, Y. and Sakurai, K.: Packet in Message Based DDoS Attack Detection in SDN Network Using OpenFlow, *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, pp. 522–528 (2017).

[13] Mininet Team: Mininet: an instant virtual network on your laptop, (online), available from ⟨http://mininet.org/⟩.

[14] Linux Foundation Collaborative Project: Open vSwitch: Production Quality, Multilayer Open Virtual Switch, (online), available from ⟨https://www.openvswitch.org/⟩.

[15] RYU project team: RYU SDN Framework, (online), available from ⟨https://osrg.github.io/ryu-book/ja/Ryubook.pdf⟩.

[16] Oracle Corporation: VirtualBox, (online), available from ⟨https://www.virtualbox.org/⟩.