OpenFlowによるフローの初期部分を対象としたミラーリングシステムの実装*

清水 貴弘[1,a)]　北川 直哉[2,b)]　山井 成良[2,c)]

概要：パケットキャプチャは，トラフィックの検査や障害発生時の原因調査に用いられる．パケットキャプチャを行う際にスイッチで設定することが多いポートミラーリングは，その特性上，スイッチに接続された全端末の双方向のトラフィックが1つのミラーポートに集中する．そのため，キャプチャされるパケットデータは非常に膨大になるほか，ミラーポートの帯域を超えてしまい，すべてのパケットのミラーリングが行えないなどの問題点がある．本論文では，OpenFlow の Packet-In の仕組みを活用することで，必要な情報の検出後はミラーリングを省略することで低負荷なパケットミラーリングを行う仕組みについて，実装手法を詳細に述べる．

Implementation of a Mirroring System for Initial Sequence of Flow Using OpenFlow

Takahiro Shimizu[1,a)]　Naoya Kitagawa[2,b)]　Nariyoshi Yamai[2,c)]

Abstract: Packet capture is used for traffic inspection and investigation of the failure occurrences. When using packet capture, an administrator configures a port mirroring by the switch in many cases. Port mirroring has a characteristic that the bidirectional traffic of all the terminals connected to its switch is concentrated on the single mirror port. Hence it has a problem that captured packet data become enormous and the huge amount of traffic exceeds the bandwidth of the mirror port. In this paper, we propose a low load packet mirroring method that abbreviates the mirroring after detecting the necessary information using "Packet-In" mechanism of OpenFlow.

1. はじめに

パケットキャプチャは，ネットワークの問題解決手法の一つであり，トラフィックや障害発生時の原因調査に用いられる．パケットキャプチャを行う際にスイッチに設定することの多いポートミラーリングでは，その特性上，スイッチに接続された全端末の双方向のトラフィックが1つのミラーポートに集中する．そのため，キャプチャされるパケットデータは非常に膨大になるほか，ミラーポートの帯域を超えてしまいすべてのパケットのミラーリングが行えないなどの問題点がある．簡便な対策手法として選択的なポートミラーリング機能 *2 を持つネットワークスイッチ [2] が存在するが，設定したフィルタリング条件にパケットが集中するような環境では効果を発揮することができず，通常のポートミラーリングと同様の問題を抱えている．この問題を解決するために，我々の研究グループでは OpenFlow の Packet-In の仕組みを活用することで，レイヤ7ヘッダのネットワーク管理者が求める情報を探し，必要な情報の発見後はミラーリングを省略することで低負荷なパケットミラーリングを行う基本的な仕組みを提案した [1]．本論文では，文献 [1] で提案した必要なパケットを

[1] 東京農工大学工学部情報工学科
Department of Computer and Information Sciences, Tokyo University of Agriculture and Technology
[2] 東京農工大学大学院工学研究院先端情報科学部門
Division of Advanced Information Technology & Computer Science, Institute of Engineering, Tokyo University of Agriculture and Technology
a) tshimizu@st.go.tuat.ac.jp
b) nakit@cc.tuat.ac.jp
c) nyamai@cc.tuat.ac.jp
*1 本論文は，文献 [1] を発展させ，その実装についてより詳細に述べたものである．

レイヤ 7 ヘッダ情報を用いて判別し，必要なパケットのみをミラーリングするシステムについて，より具体的な仕組みと実装について述べる．

2. 関連技術

本節では，本論文で取り扱う OpenFlow とポートミラーリングについて述べる．また，具体的な実装例で用いる HTTP 通信の Content-Length ヘッダについて述べる．その他，関連研究について述べる．

2.1 OpenFlow

OpenFlow[3] は，ONF(Open Network Foundation) が提唱する，SDN(Software Defined Network) を実現する技術の 1 つである．OpenFlow で実現される SDN は，従来のネットワークと異なり，ルータやスイッチの設定を各ネットワーク管理者の要求に合わせて動的に変更することができる．OpenFlow は，スイッチに事前に定義されたマッチフィールドと呼ばれる制御条件と，アクションと呼ばれる条件適合時のルーティング，スイッチング，パケットの書き換えや破棄等の動作からなるフローエントリに従って動作する．フローエントリは，事前に OpenFlow に対応したスイッチである OpenFlow スイッチに書き込むことができるほか，OpenFlow コントローラと呼ばれるプログラムによって，フローエントリを動的に変化させることができる．OpenFlow では，OpenFlow を制御するコントロールプレーンとパケットを転送するデータプレーンが分離しており，経路の制御は OpenFlow コントローラが行い，パケットの転送は OpenFlow スイッチが行う．

コントローラで動的にスイッチを制御する際は，Packet-In と呼ばれる未知パケットを OpenFlow スイッチから OpenFlow コントローラに渡す OpenFlow メッセージを利用して，パケットを OpenFlow コントローラに転送し，そのパケットの情報を基に Flow-Mod メッセージと呼ばれるフローエントリの追加や削除，および変更を行うメッセージを OpenFlow コントローラから OpenFlow スイッチへの送信により，フローエントリを追加する．また，Packet-In で送信されるパケットに対しては，Packet-Out と呼ばれる，OpenFlow コントローラから直接パケットを転送するメッセージを用いて転送する．なお，OpenFlow には様々なバージョンが存在し，それぞれ実装されている機能が異なる．

2.1.1 マッチフィールド

OpenFlow では，フローエントリとして指定できるマッチフィールドがバージョンごとに仕様としてまとめられており，これ以外のマッチフィールドを用いることは原則としてできない．そのため，レイヤ 7 の情報を参照したい場合はその情報を持ったパケットの Packet-In を確実に実施する必要がある．最も基本的な機能で構成される OpenFlow1.0[4] で設定できるマッチフィールドには，受信物理ポート番号，MAC アドレス，VLAN ID，IP アドレス，TCP および UDP の送信元ポート番号と宛先ポート番号などがある．

2.1.2 統計情報

OpenFlow スイッチは，テーブル単位や，フローエントリ単位で統計情報を持つ．統計情報は，Flow-Stats Request メッセージを，OpenFlow コントローラが OpenFlow スイッチに送信することで，OpenFlow スイッチから Flow-Stats Reply メッセージとして OpenFlow コントローラに返信される．これらのメッセージを OpenFlow コントローラと OpenFlow スイッチとの間で送受信することで，統計情報を OpenFlow コントローラで利用することができる．具体的には，フローエントリごとであれば，フローエントリにマッチしたパケットの数 (Packet Count) や総データサイズ (Byte Count) およびフローエントリが作成されてからの経過時間 (Duration) を取得することができる．

ただし，現在の最新バージョンである OpenFlow1.5[5] を用いても，統計情報をマッチフィールドとするフローエントリは設定することは出来ない．

2.2 一般的なポートミラーリング

ポートミラーリング [6] とは，スイッチやルータが持つ機能の一つで，あるポートが受信したパケットを同時に他のポートに送信する機能である．**図 1** に，クライアントとサーバとの間の通信における一般的なミラーリングの例を示す．図 1 に示すように，スイッチにあらかじめ設定したミラーポートに接続されている監視端末に対して，サーバに送っているデータと同じものをミラーリングして転送する．なお，クライアントとサーバとの間の通信も同様にミラーリングされ，監視端末に送信される．このため，ポートミラーリングを行った場合，通常の 2 倍の通信が発生することになる．さらに，接続されている機器が多くなると，それに比例してミラーリングされるパケットの量がさらに増えるため，IDS(Intrusion Detection System) 等の機器を接続した際に，ミラーポートの帯域不足でミラーリングされないパケットが発生し，障害発生の原因となる．

2.3 選択的なポートミラーリング

2.2 節で述べた一般的なポートミラーリングでは，あるポートで受信したすべてのパケットについてミラーリングを行うため，ミラーポートの帯域が不足することがある．この問題を緩和するため，ACL(Access Control List) フィルタリングを用いて，特定の条件に合致するパケットのみを選択的にミラーリングする機能を持ったネットワークス

[*2] Cisco Catalyst 4500 シリーズでは，SPAN というパケットミラーリング機能に ACL(Access Control List) を組み合わせることにより，選択的なポートミラーリングを実現できる．

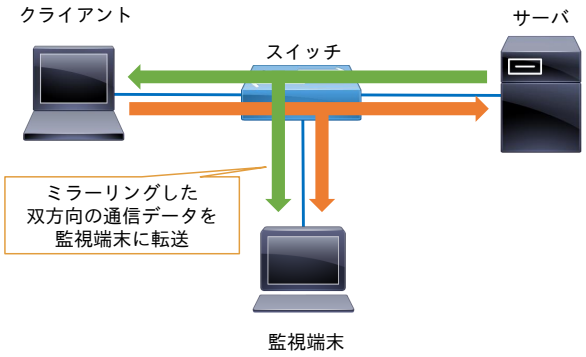

図 1　一般的なポートミラーリングの例

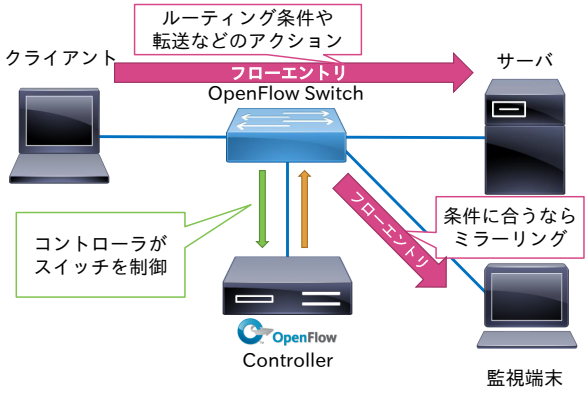

図 2　OpenFlow による選択的なポートミラーリングの例

イッチ [2] が存在する．

このようなネットワークスイッチによる選択的なポートミラーリングは，OpenFlow を用いることでも実現ができる [7]．OpenFlow コントローラで Packet-In が発生した際に，そのパケットがあらかじめ設定した条件に合致する場合，ミラーリングを行うためのフローエントリを設定する．また，パケットが条件に合致しなかった場合，通常のスイッチングのみを行うフローエントリを定義することで，フィルタリング条件にマッチする通信のみを抽出した選択的なポートミラーリングを実現できる．

しかし，ネットワークスイッチに設定するパケットフィルタリングの通過条件がスイッチを通過するパケットの大部分を占める場合，一般的なポートミラーリングと同様にミラーポートの帯域不足に陥り，すべてのパケットのミラーリングを行えない可能性がある．

2.4　HTTP 通信における Content-Length ヘッダ

HTTP(Hypertext Transfer Protocol) は，Web ブラウザなどで通信を行うときに利用されるテキストベースの通信方式である．現在，一般的に利用されている HTTP/1.1 は RFC 7230 から 7235 で定義されている通信方式であり，RFC 7230[8] において，HTTP のメッセージの構文について示されている．HTTP では，クライアントとサーバとの間でリクエストとレスポンスと呼ばれるメッセージを送受信することで，通信を実現する．また，HTTP 通信はリクエストとレスポンスに対して，HTTP ヘッダとして通信の概要が提供される．クライアントからの GET メソッドによる呼び出しに対するサーバからのレスポンスとして，送信されるデータの大きさを示す Content-Length ヘッダが応答される．Content-Length ヘッダは，同一 TCP コネクションを再利用する HTTP/1.1 の場合，通信の区切りを認識するために，サーバはレスポンスとして Content-Length ヘッダを応答する必要がある．ただし，リクエストに Content-Range が指定されている場合はこの限りではない．

2.5　関連研究

2.5.1　OpenFlow を用いたモニタリングシステム

OpenFlow を用いてネットワークのパケットを解析する手法として，OpenFlow ネットワークモニタリングシステム [9] がある．この手法は，OpenFlow コントローラと OpenFlow スイッチの間にプロキシとして動作するシステムを作成し，OpenFlow コントローラの作成した Flow-Mod メッセージを改変することで，パケットの収集およびトラフィック等の情報を収集することができる．しかし，この手法は，すべての通信シーケンスを解析の対象としているため，通信のパケット数が増えると，すべてのパケットをキャプチャできなくなるという問題がある．

2.5.2　Service-oriented Router

レイヤ 7 の情報を使用したルーティングを行うシステムとして，Service-oriented Router[10] がある．OpenFlow がレイヤ 4 までしか対応できないのに対し，このシステムでは，レイヤ 7 の情報の利用を可能にしている．ただし，この手法は，OpenFlow とは別の SDN 環境を使用して実現しているため，OpenFlow などの主要な SDN の資源を用いることができない．

3.　OpenFlow を用いた省略ミラーリングシステム

3.1　実現方針

2.2 節で述べたように，通常のポートミラーリングでは，すべてのパケットをミラーリングするため，ミラーポートの帯域が不足することによって，パケットの一部を取得できない可能性がある．また，同様に，2.3 節で述べた選択的なポートミラーリングを実施した場合でも，設定した条件にパケットが集中した場合に，従来のポートミラーリングと同様にミラーポートの帯域が不足し，パケットの一部が取得できない可能性がある．

そこで，我々は OpenFlow を用いてパケットの内容を調べ，以降のパケットの解析に必要なレイヤ 7 のヘッダ情報の取得が終わった時点でミラーリングを停止することによ

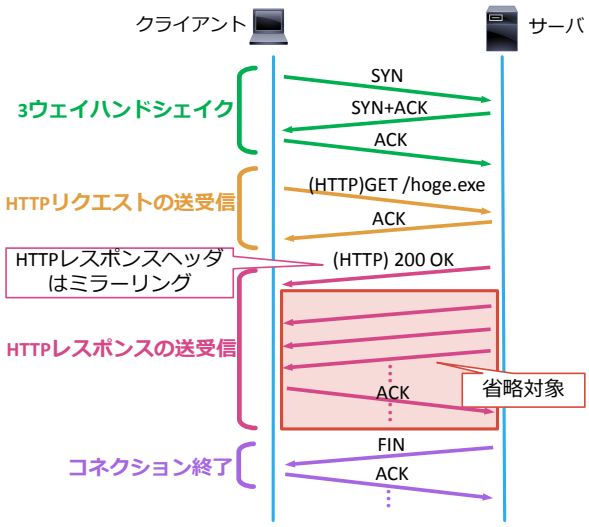

図 3　HTTP 通信において省略対象となるパケット

り，不要なパケットのミラーリングを防ぎ，ミラーポートの帯域の利用を大幅に減らすシステムを提案した．

なお，本システムに省略対象となる通信は大容量の通信のレイヤ 7 のペイロード部分にあたるパケットである．具体例として，クライアントとサーバとの間での HTTP 通信において省略対象となるパケットを，図 3 に示す．HTTP 通信においてパケットを省略する際には，HTTP 通信のレスポンスヘッダは残し，それ以外のペイロード部分を省略対象とする．他のプロトコルについても同様の考え方で省略を行うことができる．

本節では，本システム [1] の基本的な動作である，タイムアウトを用いた省略ミラーリングシステムの動作について述べる．

3.2　システムの構成

本システムの構成を図 4 に示す．本システムは OpenFlow コントローラ，OpenFlow スイッチ，クライアント，サーバおよび監視端末から構成され，監視端末に対してミラーリングしたデータを送信する．本システムの OpenFlow スイッチは，監視端末以外に対しては一般的なレイヤ 2 スイッチと同様に，MAC アドレスの学習とスイッチングを行う．監視端末に対しては，3.3 節で示す手順によって必要なパケットのみをミラーリングして送信する．OpenFlow コントローラは，OpenFlow スイッチを通過するパケットを受け取り，必要なレイヤ 7 のヘッダにあたるパケットが通過したかを判定するほか，その通信のデータ量をレイヤ 7 のヘッダから読み取る．

3.3　提案システムの動作

提案システムは，大容量の通信について以下に示す流れで動作し，ミラーリングの省略を行う [1]．

(1) 必要なパケットを OpenFlow コントローラで Packet-In

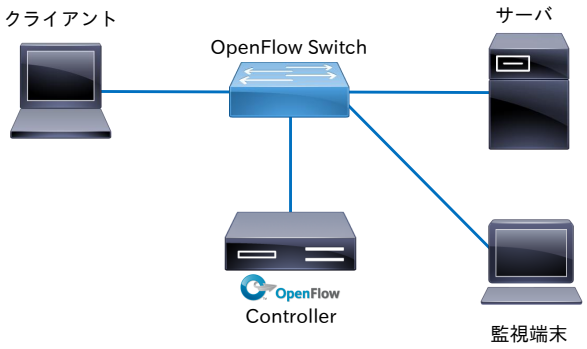

図 4　提案システムの構成

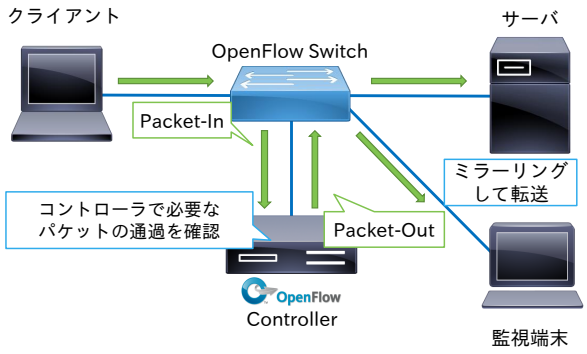

図 5　解析に必要なパラメータを見つけるまでの通信

および Packet-Out を用いて探す．

(2) 必要なパケットパラメータを発見後，フローエントリを作成し，ミラーリングを停止する．

(3) 省略対象の通信の終了を調べる．

(4) 省略対象の通信が終了後，フローエントリを削除し，ミラーリングを再開させる．以降，(1) から (4) の動作を繰り返す．

各動作について，3.3.1 節から 3.3.4 節でより詳細に述べる．

3.3.1　解析に必要なパラメータを見つけるまでの通信

図 5 に，解析に必要なレイヤ 7 の情報を見つけるまでの通信の様子を示す．クライアントから送られてきたパケットは必ず Packet-In を発生させ，OpenFlow コントローラでパケットを調べる．同様に，サーバから送られてきたパケットについても，必ず Packet-In が発生する．解析に必要な情報を含むパケットを見つけた後は，Packet-Out メッセージで正規の宛先ポートとミラーポートの双方に対してパケットを送出することにより，ミラーリングを実現する．

OpenFlow では，Packet-Out のタイミングでフローエントリを書き込むのが一般的であるが，本システムでは対象の通信について解析に必要なデータが見つかるまでは Flow-Mod メッセージでフローエントリの書き込みを実施しない．なお，対象とする通信以外は通常のポートミラーリングを行うフローエントリを作成し，必要以上の Packet-In を発生させないものとする．

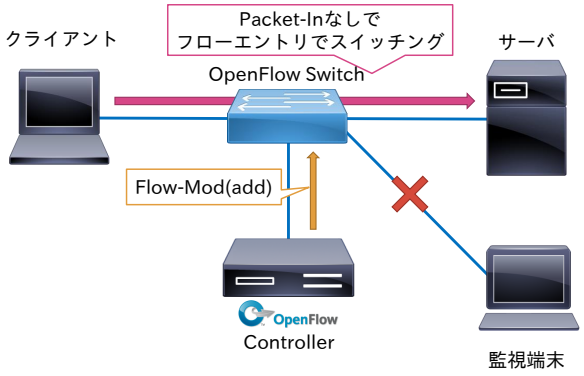

図 6　ミラーリングの停止

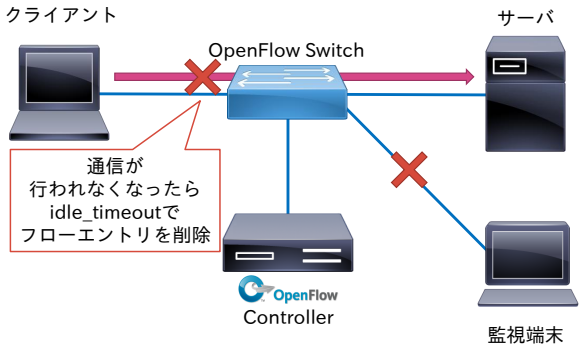

図 7　ミラーリング停止中の処理

3.3.2　ミラーリングの停止

図 6 に，ミラーリング停止時の通信の様子を示す．3.3.1 節で述べた Packet-In によるパケットの検査によって必要なデータを発見できたら，フローエントリを書き込む Flow-Mod メッセージを送出する．書き込むフローエントリは，TCP ポート番号と IP アドレス，MAC アドレスなどの通信相手の対応を特定できるものをマッチフィールドの条件として設定し，正規の宛先ポートに対してスイッチングを行うアクションを指定してフローエントリを書き込む．これによって当該の通信に関する Packet-In が停止し，それと同時にミラーリングも停止する．

3.3.3　ミラーリング停止中の処理

図 7 に，ミラーリング停止中の通信の様子を示す．

ミラーリング停止中は，OpenFlow コントローラと OpenFlow スイッチ間での Packet-In および Packet-Out は停止し，フローエントリを基にサーバとクライアント間でのみ通信を行う．コネクションが再利用されないプロトコルについては，フローエントリが参照されない時間が一定時間続いたときに削除を行う idle_timeout をフローエントリに設定することでフローエントリを削除するため，ミラーリング停止中に OpenFlow コントローラによる監視を行う必要はない．一方でコネクションを再利用するプロトコルの場合には，フローエントリの統計情報の監視などの処理を加える必要がある．この点について 4.2.3 節で詳述する．

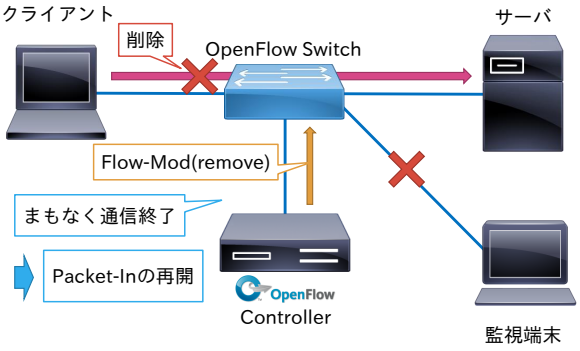

図 8　ミラーリングの再開

3.3.4　ミラーリングの再開

図 8 に，ミラーリング再開時の通信の様子を示す．ミラーリング停止中に，統計情報を監視している時に，通信が終了する直前となった時点で，Packet-In を再開するために，Flow-Mod メッセージを用いてフローエントリを削除する．これにより，Packet-In が再開され，再度 OpenFlow コントローラを必ず経由する形でスイッチングが行うことができる状態になる．Packet-In が再開した時点で当該パケットを調べ（3.3.1 節で詳述），その結果から Packet-Out メッセージで正規の宛先ポートとミラーポートにパケットを送出することにより，再度ミラーリングを行う．以後，3.3.1 節から 3.3.4 節の動作を繰り返し実行する．

4.　システムの実装

本節では，3.1 節で示した HTTP 通信を例に，具体的な実装について示す．

4.1　実装環境

本システムは，1 台のコンピュータ上の仮想環境を用いて，図 4 に示した構成を実現した．仮想環境による構成を図 9 に，仮想マシンの構成を表 1 に示す．

仮想環境のネットワークは，図 9 に示すように，OpenFlow スイッチと OpenFlow コントローラ間の接続をループバックで接続するほか，OpenFlow スイッチとコントローラ間を Host-Only ネットワークで接続している．また，OpenFlow スイッチとサーバと監視端末の接続に関しては，それぞれ別の内部ネットワークを使用して，図 4 の環境を実現した．なお，仮想環境のハイパーバイザとして Oracle VirtualBox 5.0.2 を使用し，ホストマシンに Intel Core i7-4500U(1.80GHz)，メモリ 8GB の Windows8.1 を搭載したコンピュータを使用した．仮想 NIC には，ハイパーバイザに用意されている Intel PRO/1000 MT Desktop(82540EM) を使用した．

実装環境では，仮想マシンの台数を極力減らすため，OpenFlow スイッチである Open vSwitch[11] と，Ruby 製の OpenFlow コントローラである Trema[12] を同一の仮想マシン上で動作させ，ループバックインタフェースを用い

表 1　実装環境の概要

	コンピュータ名	OS	ソフトウェア名
OpenFlow コントローラ	ゲスト OS(VM1)	Ubuntu Server 14.04(64bit)	Trema 0.4.7
OpenFlow スイッチ	ゲスト OS(VM1)	Ubuntu Server 14.04(64bit)	Open vSwitch 2.0.2
クライアント	ホスト OS	Windows8.1 Pro(64bit)	Internet Explorer 11
サーバ	ゲスト OS(VM2)	Ubuntu Server 14.04(64bit)	Apache 2.4.7
監視端末	ゲスト OS(VM3)	Ubuntu Desktop 14.04(64bit)	Wireshark 1.12.10

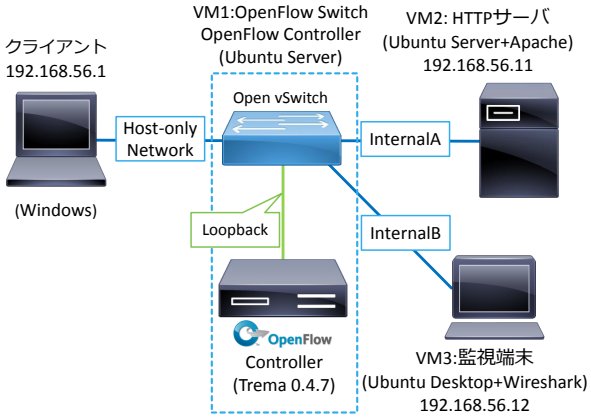

図 9　仮想環境の構成

て OpenFlow のコントロールプレーンを作成した．2.1 節で述べたように，OpenFlow には異なる機能を持つさまざまなバージョンが存在するが，本実装では最も基本的な構成である OpenFlow1.0[4] を使用した．本システムでは，サーバのポートが TCP の 80 番ポートの通信については，HTTP 通信であるとして処理する．

4.2　実装方法

本節では，OpenFlow コントローラと OpenFlow スイッチの動作について，実装上注意が必要な点を述べる．

4.2.1　Packet-In 時の動作

Packet-In 時には，OpenFlow スイッチは通常のスイッチとしての物理ポートの学習を行う．また，OpenFlow コントローラではパケットの確認を実施する．また，OpenFlow のフローエントリは，単方向の転送情報のみを持つ．そのため，OpenFlow コントローラで Packet-In 時の処理を行う際には，通信が双方向のものであることに留意して実装する必要がある．

OpenFlow には，レイヤ 7 の情報を解析する機能はないため，OpenFlow コントローラで独自で実装する必要がある．本実装では，TCP パケットの解析には，OpenFlow コントローラ開発フレームワークである Trema に付属するパケット解析および生成を行うライブラリである Trema::Pio を使用した．Trema::Pio には，TCP パケットを解析する機能はないため，TCP 通信を解析するために機能拡張を行った．拡張した Trema::Pio で TCP パケットのペイロード部分を取り出した後は，レイヤ 7 の情報である HTTP 通信の内容を正規表現を用いて解析する．具体的には，パケットのペイロード先頭について，HTTP リクエストヘッダが含まれていることを確認し，含まれている場合に限りパラメータを取得する処理を行う．これにより，不要なパケット解析の負荷を抑えることができる．

なお，本システムでは，HTTP 通信のレスポンスメッセージに含まれる Content-Length パラメータを発見した後，ミラーリングの省略を実施する．Content-Length パラメータの値は，3.3.3 節で述べた通信終了時間の推定に使用するため，パケットの情報としてクライアント側 IP アドレス，サーバ側 IP アドレスおよび TCP のクライアント側ポート番号をキーとするハッシュテーブル内に格納し，以降の処理で使用できるようにする．

4.2.2　コントロールプレーンの遅延

OpenFlow コントローラと OpenFlow スイッチが接続されるコントロールプレーンでは，ネットワークで接続されている以上，遅延を考慮する必要がある．HTML ファイル等のファイルサイズの小さいデータについてペイロードの省略を試みると，Flow-Mod メッセージでフローエントリの書き込みおよび削除，統計情報を取得などに要する時間の方が長いことがある．このため，パケットのミラーリングの省略を実施を試みたときには既に通信が終わっているため，意図しない動作をする可能性がある．

また，通信データサイズが小さい場合は，通信がすぐに終了するためミラーポートの帯域への影響は小さいため，ミラーリングするパケットの省略は事実上不要である．従って，本システムでは 10MB 以上のデータ通信が発生しない場合はフローエントリの書き込み等の動作を行わず，Packet-In および Packet-Out メッセージですべてのパケットを処理し，すべてミラーリングする実装とした．一方，10MB 以上のデータ通信が発生する場合については，ミラーポートの帯域圧迫を防止するために，ミラーリングするパケットの省略を実施する．

4.2.3　コネクションを再利用するプロトコル

TCP コネクションを再利用するプロトコルの場合，通信終了を検知するためにフローエントリの統計情報の監視が必要である．例えば，HTTP/1.1 では，1 回の TCP コネクションで複数の HTTP リクエストを処理する HTTP Keep Alive がサポートしている．統計情報の監視は，**図 10** に示すように，ミラーリング停止中は，OpenFlow コントローラから OpenFlow スイッチに対して，統計情報を取得す

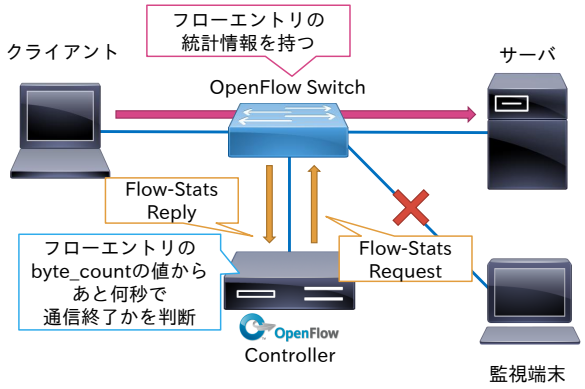

図 10　コネクションの再利用を行う場合の通信の監視

る Flow-Stats Request メッセージを随時送信する．その結果として OpenFlow スイッチから送信される Flow-Stats Reply メッセージを受信し，フローエントリに該当したパケットの数とデータ量を常に監視する．これらの処理で OpenFlow スイッチから取得した統計情報と，通信の初期に Packet-In を用いて取得した通信データ量を比較して，ミラーリングの復帰に必要な通信の終了時間を推定する．

5. 本システムによるミラーリングパケットの削減効果

本節では，OpenFlow を用いて通常のポートミラーリングを行った結果と，本システムを用いてミラーリングの省略を行った結果 [1] について，より詳細に述べる．なお，通常のポートミラーリングを行う環境は，4.1 節で述べた本システムと同一の環境で別途実装した．

本評価では，動作例として，クライアントがサーバから約 36MB のバイナリファイルを HTTP 通信の GET リクエストによりダウンロードする通信を発生させた．**表 2** に，ミラーリングされたパケットの数とデータ量を示す．また，**図 11** と**図 12** に，実際にパケットキャプチャツールである Wireshark[13] を用いた監視端末におけるパケットキャプチャの結果を示す．図 11 と図 12 のいずれも，HTTP 通信についてのパケットのみを掲載した．

図 11 は，本システムを用いたパケットの省略を実施せず，通常のポートミラーリングを行った時のパケットキャプチャの結果の一部を示す．ただし，実際にキャプチャされたパケット数は，表 2 に示すように 7,418 パケットである．また，約 36MB の通信を発生させたにも関わらず，表 2 に示すように，パケットの省略を実施しない場合のミラーリングデータ量が約 10MB となった．これは，通常のポートミラーリングでは帯域が不足し，多くのパケットのミラーリングに失敗したためである．

図 12 は，本システムを用いてパケットの省略を実施した時のパケットキャプチャの結果であり，対象とする HTTP 通信について受信した 13 パケットすべてを示している [1]．図 12 中において，TCP 通信におけるシーケンス番号と確

表 2　システムの動作結果 [1]

	パケット数 [個]	データ量 [Byte]
省略なし	7,418	10,444,377
省略あり	13	9,927

認応答番号が合わないために一部のパケットについて警告が出力されている．これは，本システムによって TCP 通信の一部が省略されたために出力されたものであり，本システムの正常な動作を示すものである．

約 36MB のファイルの転送において，本システムによる省略後のミラーリングパケットのデータ量は表 2 に示すように 9,927 バイトであり，約 35MB の削減を観測した．

6. 考察

実装について，省略を行うべき通信の大きさの閾値は，現段階の実装では 10MB としているが，この値は本システムを使用する環境に依存するため，環境のスループットに応じて変化させる必要がある．4.2.3 節で示したようなコネクションを再利用するプロトコルの実装については，通信終了時間の推定精度によっては Packet-In の再開が早過ぎると，スループットに影響を及ぼす可能性が考えられる．また，OpenFlow では再送を検知する方法がないため，本論文で述べた実装は再送の多発する環境において通信終了時間の推定精度に悪影響が出ることが予測される．

パケットの削減効果について，パケットの省略を実施せずに通常のポートミラーリングを実施した場合には帯域不足の状態に陥ったが，本システムを使用してパケットの省略を実施した場合，表 2 に示すように大幅に転送データ量を削減することができることから，高い負荷耐性があるといえる．また，本実験では 1 台のコンピュータ上の仮想環境を用いたが，実運用時には現在流通している一般的な OpenFlow ネットワーク用機器を使用することにより，実運用環境に適した運用を行うことが可能であると言える．

本システムは，ネットワーク型 IDS(Intrusion Detection System) のような，継続してパケットを収集するシステムにおいて効果があると予測される．ミラーリングパケットの量の減少による負荷軽減効果や，レイヤ 7 ヘッダ部分のミラーリングの可能性の向上による解析精度の向上などの効果が期待できる．

一方，本実装ではパラメータが見つかるまで Packet-In 経由でミラーリングを実施するため，Packet-In 処理の遅延が発生するとボトルネックとなる．現在は実装に Ruby を用いている他，正規表現を多用しているため，処理方法の改善により，スループットの向上が期待できる．

7. おわりに

本論文では，ポートミラーリングにおける帯域不足を解消するため，通信のうち，レイヤ 7 のヘッダ部分のみをミ

Time	Source	Destination	Protocol	Length	Info
101.113309	192.168.56.1	192.168.56.11	TCP	66	59299→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
101.121697	192.168.56.11	192.168.56.1	TCP	66	80→59299 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
101.122612	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
101.126998	192.168.56.1	192.168.56.11	HTTP	459	GET /Wireshark-win32-2.0.0.exe HTTP/1.1
101.127037	192.168.56.11	192.168.56.1	TCP	60	80→59299 [ACK] Seq=1 Ack=406 Win=30336 Len=0
101.129145	192.168.56.11	192.168.56.1	TCP	14654	[TCP segment of a reassembled PDU]
101.131367	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=406 Ack=1461 Win=65536 Len=0
101.131398	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=406 Ack=2921 Win=65536 Len=0
101.131409	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=406 Ack=4381 Win=65536 Len=0
101.131430	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=406 Ack=5841 Win=65536 Len=0
101.131447	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=406 Ack=7301 Win=65536 Len=0
101.131458	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=406 Ack=8761 Win=65536 Len=0
101.131480	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=406 Ack=10221 Win=65536 Len=0
101.131492	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=406 Ack=11681 Win=65536 Len=0
101.132125	192.168.56.11	192.168.56.1	TCP	5894	[TCP segment of a reassembled PDU]
101.132334	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=406 Ack=13141 Win=65536 Len=0
101.132348	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=406 Ack=14601 Win=65536 Len=0
101.132358	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=406 Ack=16061 Win=65536 Len=0
101.132368	192.168.56.1	192.168.56.11	TCP	60	59299→80 [ACK] Seq=406 Ack=17521 Win=65536 Len=0

図 11 省略なしの場合のパケットキャプチャ結果 (一部)

Time	Source	Destination	Protocol	Length	Info
0.00000000	192.168.56.1	192.168.56.11	TCP	66	53523→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
0.06823500	192.168.56.11	192.168.56.1	TCP	66	80→53523 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
0.07718700	192.168.56.1	192.168.56.11	TCP	60	53523→80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
0.08597200	192.168.56.1	192.168.56.11	HTTP	459	GET /Wireshark-win32-2.0.0.exe HTTP/1.1
0.32225900	192.168.56.11	192.168.56.1	TCP	60	80→53523 [ACK] Seq=1 Ack=406 Win=30336 Len=0
0.32258900	192.168.56.11	192.168.56.1	TCP	1514	[TCP segment of a reassembled PDU]
0.32395800	192.168.56.11	192.168.56.1	TCP	1514	[TCP segment of a reassembled PDU]
0.33497200	192.168.56.11	192.168.56.1	TCP	1514	[TCP segment of a reassembled PDU]
0.98072900	192.168.56.11	192.168.56.1	TCP	4434	[TCP segment of a reassembled PDU]
8.66872100	192.168.56.11	192.168.56.1	TCP	60	[TCP Previous segment not captured] 80→53523 [FIN, ACK] Seq=37999688 Ack=406
8.67764700	192.168.56.1	192.168.56.11	TCP	60	[TCP ACKed unseen segment] 53523→80 [ACK] Seq=406 Ack=37999688 Win=6883584 L
8.67795600	192.168.56.1	192.168.56.11	TCP	60	53523→80 [ACK] Seq=406 Ack=37999689 Win=6883584 Len=0
12.6780030	192.168.56.1	192.168.56.11	TCP	60	53523→80 [RST, ACK] Seq=406 Ack=37999689 Win=0 Len=0

図 12 省略ありの場合のパケットキャプチャ結果 [1]

ラーリングし，大部分のパケットのミラーリングを省略することにより，ミラーポートの帯域を圧迫するトラフィックを大幅に削減するシステムの詳細な実装について述べた．本論文では仮想環境上の簡易な実装について述べたが，実用化に向けた機器の選択や構成は今後検討する必要がある．今後の課題として，実運用環境での検証のほか，SSL 通信などの暗号化された通信に対するパケットの省略方法の検討が挙げられる．SSL 通信については，OpenFlow コントローラで SSL インスペクションにより対応可能である他，リバースプロキシを用いた SSL 環境では，内部ネットワークに対して本システムを適用するなど，運用の形態に合わせて SSL 通信の対応方法を検討する必要がある．

参考文献

[1] 清水貴弘，山井成良，北川直哉：OpenFlow を用いたレイヤ 7 ペイロードの省略による低負荷なパケットミラーリングシステムの実装，第 15 回情報科学技術フォーラム講演論文集第 4 分冊，pp. 133–134 (2016).

[2] シスコシステムズ：Cisco Catalyst 4500 シリーズ（オンライン），入手先 ⟨http://www.cisco.com/web/JP/product/hs/switches/cat4500/index.html⟩（参照 2016-09-12）.

[3] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S. and Turner, J.: OpenFlow: Enabling Innovation in Campus Networks, *SIGCOMM Comput. Commun. Rev.*, Vol. 38, No. 2, pp. 69–74 (online), DOI: 10.1145/1355734.1355746 (2008).

[4] Open Networking Foundation: OpenFlow Switch Specification Version 1.0.0 (online), available from ⟨https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf⟩ (accessed 2016-09-12).

[5] Open Networking Foundation: OpenFlow Switch Specification Version 1.5.0 (online), available from ⟨https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf⟩ (accessed 2016-09-12).

[6] 井上孝司：ゼロから始めるスイッチ講座 (6) ポートミラーリングの設定（オンライン），入手先 ⟨http://news.mynavi.jp/series/networkswitch/006/⟩（参照 2016-09-12）.

[7] あきみち，宮永直樹， 岩田淳：マスタリング TCP/IP OpenFlow 編，chapter 選択的ポートミラーリング, pp. 130–132，オーム社 (2013).

[8] Fielding, R. T. and Reschke, J. F.: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, RFC 7230 (2015).

[9] 堤啓彰，谷口義明，井口信和，渡辺健次：OpenFlow ネットワークモニタリングシステムの開発，インターネットと運用技術シンポジウム 2014 論文集，Vol. 2014, pp. 23–30 (2014).

[10] Takagiwa, K., Ishida, S. and Nishi, H.: SoR-Based Programmable Network for Future Software-Defined Network, *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, pp. 165–166 (online), DOI: 10.1109/COMPSAC.2013.29 (2013).

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

[12] Trema: Trema (online), available from ⟨http://trema.github.io/trema/⟩ (accessed 2016-09-12).

[13] Wireshark Foundation: Wireshark (online), available from ⟨https://www.wireshark.org/⟩ (accessed 2016-09-12).