VM 内情報を利用する仮想P4スイッチの安全な実行

岩井 正輝[1]　光来 健一[1]

概要：近年，ネットワークスイッチにおいて P4 言語を用いてパケットの転送処理をプログラミングできるようになってきている．仮想マシン (VM) を接続する仮想スイッチの場合，P4 プログラムはホストの管理者によってロードされる．VM のユーザが仮想スイッチに P4 プログラムをロードできれば，VM ごとに VM 内の情報を用いてきめ細かいパケット処理が可能となる．しかし，クラウドの仮想スイッチは信頼できるとは限らず，ユーザの P4 プログラムも異常な動作をする可能性がある．そこで本稿では，VM 内情報を利用できるように拡張した P4 プログラムをユーザごとに用意される P4 VM で安全に実行する P4 Shield を提案する．P4 Shield は AMD SEV などを用いて P4 VM をクラウドから保護し，P4 プログラムを P4 VM に隔離することで仮想スイッチを保護する．P4 プログラムは uBPF バイトコードにコンパイルして実行し，共有メモリ上に格納されたユーザの VM 内の情報を新たに定義した外部関数を用いて取得する．P4 Shield を Open vSwitch と uBPF ランタイムに実装し，動作の確認および性能の測定を行った．

1. はじめに

近年，ネットワークスイッチにおいて P4 言語 [4] を用いてパケットの転送処理をプログラミングできるようになってきている．例えば，高度なパケットフィルタリングを行ったり，パケットのデータ書き換えを行ったりすることができる．P4 により，対応するネットワークスイッチの登場を待つことなく，新たな技術を利用することが可能となる．クラウドにおいて仮想マシン (VM) をネットワークに接続する際には仮想スイッチが用いられるが，P4 に対応した仮想スイッチも開発されている [15]．

P4 プログラムは管理者によって仮想 P4 スイッチにロードされるが，ユーザが独自の P4 プログラムをロードできるようになれば，VM ごとに柔軟なパケット転送処理が実現できるようになる．加えて，P4 プログラムの中で VM 内の情報も利用できれば，よりきめ細かいパケット処理が可能となる．しかし，クラウドによって提供されている仮想スイッチは信頼できるとは限らないため，ユーザの P4 プログラムを改ざんされたり，P4 プログラムが取得した VM 内情報を盗聴されたりするリスクがある．逆に，ユーザの P4 プログラムの挙動が仮想スイッチに影響を与える恐れもある．

そこで本稿では，VM 内情報を利用できるように拡張した P4 プログラムをユーザごとに用意される VM (P4 VM) で安全に実行する P4 Shield を提案する．P4 Shield では，仮想スイッチが受信したパケットを P4 VM に転送して P4 プログラムを実行し，実行結果に基づいてパケットの転送処理を行う．P4 Shield は AMD SEV などを用いて P4 VM を保護することにより，クラウドによる P4 プログラムへの攻撃を防ぐ．また，P4 プログラムを P4 VM に隔離して実行することにより，P4 プログラムの異常な動作から仮想スイッチを保護する．

P4 Shield を Open vSwitch 3.2.1 [2] と uBPF ランタイム [5] に実装した．P4 Shield は P4 プログラムを uBPF バイトコードにコンパイルし，P4 VM 内で動作する uBPF ランタイムを用いて実行する．P4 プログラムからユーザの VM 内の情報を安全に取得できるようにするために，ユーザ VM は P4 VM との間でメモリの一部を共有し，暗号化した情報を格納する．そして，P4 プログラムがユーザ VM 内の情報を取得するための API を外部関数として定義し，外部関数の中で uBPF ランタイムに実装したヘルパー関数を実行することで共有メモリにアクセスする．

P4 Shield を用いて，P4 VM 内の P4 プログラムがユーザ VM 内の情報を利用してパケットフィルタリングを行えることを確認する実験を行った．その結果，ユーザ VM が送信していない UDP パケットの不着を示す ICMP パケットを受信した際に，不正なパケットとして破棄できることを確認した．また，P4 Shield における TCP および UDP のレイテンシとスループットを測定し，現在の実装上の問題により性能が大幅に低下することが分かった．

以下，2 章でユーザがクラウドの仮想スイッチに P4 プログラムをロードできるようにするための課題を示す．3 章

[1] 九州工業大学
Kyushu Institute of Technology

で VM 内情報を利用できるように拡張した P4 プログラムをユーザごとに用意される P4 VM で安全に実行する P4 Shield を提案する．4 章で P4 Shield の実装について説明し，5 章で P4 Shield の動作確認と性能測定のために行った実験について述べる．6 章で関連研究を示し，7 章で本稿をまとめる．

2. ユーザによる仮想 P4 スイッチの利用

P4 言語を用いてネットワークのデータプレーンをプログラミング可能な P4 スイッチが登場している．P4 プログラムをロードすることで P4 スイッチにおいて動的なパケットフィルタリングやパケットデータの書き換えなどを行うことが可能になる．P4 プログラムではパケットのヘッダやペイロード，スイッチのポート情報などを含むメタデータを利用することができる．例えば，SRv6 [7], [8], [9] や VXLAN [14] のほか，独自実装のアプリケーションヘッダに基づく処理も P4 プログラムの記述により容易に実現可能である．P4 スイッチを用いることで，対応するスイッチの登場を待つことなく新たな技術を利用することができる．また，今後，新たに標準化を目指すようなユースケースにおいても容易に検証を行うことができる．

VM 向けには P4rt-OVS [15] や P4-OvS [3] などの P4 対応の仮想スイッチが開発されている．仮想スイッチはホストの内部に作られる仮想的なスイッチであり，Open vSwitch (OVS) [16] がよく利用されている．仮想スイッチに VM の仮想 NIC を接続することにより，仮想ネットワークを構築することができる．さらに，仮想スイッチをホストの物理 NIC に接続することにより他のホストの内部の仮想スイッチと接続することができ，リモートの VM との通信が可能になる．VM が送受信するパケットはすべて仮想スイッチを経由するため，仮想 P4 スイッチを用いることでパケット転送時に P4 プログラムを実行することができる．

物理的な P4 スイッチではネットワーク管理者が P4 プログラムをロードするのに対し，仮想 P4 スイッチに対してはホストの管理者が P4 プログラムをロードすることができる．例えば，P4rt-OVS では ovs-ofctl コマンド，P4-OvS では ovs-p4ctl コマンドを用いて P4 プログラムがロードされる．さらに，VM のユーザが P4 プログラムをロードできるようになれば，VM ごとに異なるパケット処理を行えるようになり，より柔軟なパケット転送処理が実現できるようになる．加えて，VM と連携してその内部の情報も利用することができれば，よりきめ細かいパケット処理が可能となる．例えば，パケットを送受信するアプリケーションのプロセス名や当該プロセスの実行ユーザ情報を利用したパケットフィルタ [17] などを実現できる．

しかし，ユーザが作成したこのような P4 プログラムを仮想 P4 スイッチで実行できるようにするにはいくつかの課題がある．まず，クラウドで提供される仮想 P4 スイッチはユーザにとって信頼できるとは限らない．信頼できないクラウド管理者によって管理されている場合，ユーザがロードした P4 プログラムに対して様々な攻撃が考えられる．例えば，P4 プログラム自体や P4 プログラムが参照する VM 内情報を盗聴されると機密性が失われる危険性がある．P4 プログラムを改ざんされてユーザが意図しない処理を行われると完全性が失われる．さらに，P4 プログラムを実行しないことによって VM が正常な通信を行えなくなったりサービス妨害攻撃を防げなくなったりして，可用性が失われることもある．

逆に，ユーザがロードした P4 プログラムに意図的または意図せずに引き起こされる問題が含まれていると，仮想 P4 スイッチの挙動に大きな影響を与える恐れがある．例えば，P4 プログラムが CPU やメモリなどのリソースを大量に消費する場合，すべてのパケットの転送が遅延する可能性がある．P4 プログラムやそれを実行するランタイムに脆弱性があった場合，仮想 P4 スイッチの制御を奪われるリスクもある．実際に，P4 プログラムが無限ループに陥り以降のパケットをすべて破棄したり，宛先ポートが正しく指定されていないパケットをすべて特定のポートに差し向けたり，特定条件下において以前に転送したパケットの情報を漏洩したりするなど，様々な脆弱性が指摘されている [6]．

また，P4 プログラムから VM 内の情報を扱う必要があるが，P4 プログラムはパケット内の情報しか参照することができない．VM 内の情報を取得するには仮想 P4 スイッチが VM と通信を行うことが考えられる．しかし，パケット処理ごとに通信を行っていては，実用に耐え得る転送性能を得ることが難しくなる．そのため，VM イントロスペクション [10] を用いて VM のメモリ上の情報を直接，取得する方法が考えられる．この手法を用いれば高速化が可能になるが，近年のクラウドでは AMD SEV などによりメモリが保護された Confidential VM も提供されている．クラウド管理者は Confidential VM のメモリにアクセスできないため，悪意のあるクラウド管理者からの攻撃を防ぐことが可能となる一方，仮想 P4 スイッチもまた Confidential VM のメモリ上にある情報を取得できなくなる．

3. P4 Shield

P4 Shield は**図 1** に示すように，ユーザの VM 内の情報を利用できるように拡張した P4 プログラムを P4 のための VM (P4 VM) 内で安全に実行する．P4 Shield はユーザごとに P4 VM を用意し，仮想スイッチがそのユーザの VM に関連するパケットを受信した時に，対応する P4 VM にパケットを転送して P4 プログラムを実行する．その際に，ユーザ VM 内の情報を利用してパケットの処理を行い，パケットの転送・破棄の判定や書き換えたパケットのデータ

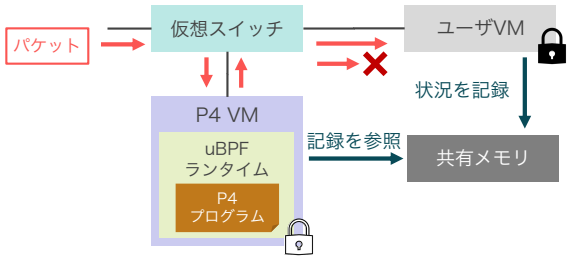

図 1: P4Shield のシステム構成

を仮想スイッチに返送する．それを基に仮想スイッチはパケットの転送処理を行う．

P4Shield は AMD SEV などを用いて P4 VM を Confidential VM として実行することにより，仮想スイッチを含むクラウド全体から保護する．P4 VM のメモリは暗号化されるため，クラウドは P4 プログラムが取得した VM 内情報を盗聴することはできない．VM のメモリの整合性が保持されるため，クラウドは P4 プログラムを改ざんすることもできない．Confidential VM を用いても可用性は保証できないため，P4Shield は後述する別の手法を用いて P4 プログラムの実行を担保する．また，従来の仮想 P4 スイッチと異なり，P4 プログラムを仮想スイッチ内で実行しないようにすることにより P4 プログラムから仮想スイッチを保護することができる．P4 プログラムが異常な動作をしたとしても影響を受けるのは P4 VM だけである．

P4 VM 内では同一ユーザの P4 プログラムが複数実行されることがあるため，uBPF [5] を用いて P4 プログラム同士を保護する．P4Shield はユーザが作成した P4 プログラムを uBPF バイトコードにコンパイルし，ロード時に JIT コンパイルすることで高速に実行する．uBPF は eBPF [1] のユーザランド実装であり，プログラムの実行前に無限ループや安全でない操作などが含まれていないかの検証が行われる．また，プログラムごとに個別の実行環境が提供されるため，互いに影響を及ぼすことがない．P4 VM 内では uBPF ランタイムが常時起動しており，仮想スイッチから転送されたパケットが届くたびにロード済みの P4 プログラムを実行する．

P4 プログラムからユーザ VM 内の情報を取得できるようにするために，P4 VM はユーザ VM のメモリの一部を共有する．ユーザ VM 内の OS がパケットに関連する情報を共有メモリに格納し，P4 VM 内の P4 プログラムがその情報を共有メモリから取得して利用する．格納する情報の例としては，通信を行っているプロセス名やそのプロセスを実行したユーザ名などが挙げられる．これにより，P4 プログラムは高速にユーザ VM 内の情報を取得することができる．ユーザ VM が Confidential VM として実行されている場合には，当該メモリ領域を暗号化しないようにする．その結果，クラウドによる盗聴・改ざんも可能となるため，共有メモリに格納する情報はユーザ VM 内で暗号化してから格納し，P4 VM での取得時に復号および整合性検査を行う．そのために必要な暗号鍵はユーザ VM と P4 VM 間で安全に共有する．

P4Shield は P4 プログラムがユーザ VM との間に確立された共有メモリにアクセスできるようにするために P4 の拡張を行う．P4 プログラムを uBPF バイトコードにコンパイルするコンパイラは，P4 プログラムによる外部関数の呼び出しをサポートしている．この機能を利用して，P4Shield は共有メモリ上にあるユーザ VM 内の情報を取得するための API を外部関数として定義する．P4 プログラムがこの外部関数を呼び出すと uBPF ランタイム内のヘルパー関数が呼び出され，uBPF ランタイムが共有メモリにアクセスする．

P4 プログラムが正常に実行されたかどうかをユーザが確認できるようにするために，P4 VM は P4 プログラムの実行ログをユーザ VM との間の共有メモリに格納する．仮想スイッチは P4 VM にパケットを転送せずにパケット処理を行うことで P4 プログラムの実行を行わないようにすることが可能である．このような攻撃を検知するために，ユーザ VM は共有メモリに格納された実行ログを定期的に取得し，パケット送受信の統計データと照合する．実行ログと統計データが一致しない場合には，P4 プログラムが正常に実行されていないことを検知することができる．クラウドによる共有メモリ上の実行ログの改ざんを防ぐために，ユーザ VM において整合性検査を行う．

4. 実装

P4Shield を OVS 3.2.1 [2] と uBPF ランタイム [5] に実装した．

4.1 OVS の拡張

OVS のユーザランドデータパスである netdev に対して P4 のための拡張を行った．OVS は起動時に netdev インタフェースを初期化し，パケット到着時に受信キューから最大で 32 個のパケットを一括取得してパケットバッチにまとめ，パケットの転送処理を行う．P4Shield ではパケットの転送処理を行う直前に P4 VM にパケットデータを送り，P4 プログラムの実行結果を受け取る．パケットデータとしては，パケットのヘッダおよびペイロード，ハッシュ値などを送る．そして，P4 VM から受信した実行結果に基づき，パケット転送処理を続行または中止する．実装を簡単にするために，1 つのパケットバッチに内含可能な最大パケット数を 1 に制限している．また，パケットデータや実行結果の転送は TCP ソケット通信を用いて実装している．今後，P4 VM との間に確立した共有メモリを用いて実装する予定であるが，現時点では未実装である．

```
extern bit<8> checkUDPSent();
if (checkUDPSent() == 1)
  mark_to_drop();
```

図 2: P4 プログラム例

4.2 uBPF ランタイムの拡張

uBPF バイトコードを実行するための uBPF ランタイムを拡張し，OVS と連携して VM 内情報を利用する P4 プログラムを実行できるようにした．P4Shield はユーザの P4 プログラムから生成された uBPF バイトコードを P4 VM で実行される uBPF ランタイムにロードする．その際に uBPF バイトコードを JIT コンパイルし，ネイティブ実行できるようにする．その後，uBPF ランタイムは OVS からパケットデータが送られてくるまで待機する．OVS からパケットデータを受信すると，そのサイズ情報を用いてメタデータを作成し，パケットデータとメタデータを uBPF バイトコードの引数にして実行を開始する．実行を終えると，その返り値を OVS に返却する．

uBPF ランタイムに P4 VM とユーザ VM との間に確立された共有メモリにアクセスしてデータを読み込むヘルパー関数 `getUDPSent()` を新たに定義した．uBPF ランタイムは様々なヘルパー関数に一意の番号を割り振って管理している．そのため，uBPF ランタイムは初期化時に新たに定義したヘルパー関数と番号との対応付けを登録する．また，P4 プログラムの実行に必要なヘルパー関数も同様に登録する．uBPF バイトコードの実行時にこれらの番号を指定した call 命令が実行されると，uBPF ランタイムは対応するヘルパー関数を実行する．

4.3 P4 の拡張

P4 プログラムからユーザ VM 内の情報を取得するための外部関数 `checkUDPSent()` を定義した．この外部関数は P4Shield の動作確認のために定義したものであり，ユーザ VM が直近に UDP パケットを送出したかどうかという情報を uBPF ランタイムに定義したヘルパー関数を呼び出すことで共有メモリから取得する．その結果に基づいて，UDP パケットを送出していれば 1，送出していなければ 0 を返す．P4 プログラムでは**図 2** のように外部関数を extern 宣言し，通常の関数のように呼び出すことで利用する．一般的には，外部関数はパケットヘッダ全体またはその中のアドレスやポート番号などを引数として受け取り，何らかの値を返り値として返す．また，引数でユーザ VM 内の情報をそのまま返すこともできる．

P4 プログラムのコンパイル時にはまず，P4 リファレンスコンパイラである p4c に含まれる uBPF 向けコンパイラの p4c-ubpf を用いて，P4 プログラムを C 言語のプログラムに変換する．コンパイラを修正することにより，新たに追加したヘルパー関数の名前を持つ関数ポインタを定義し，ヘルパー関数に割り当てられた番号を void *型にキャストして代入するコードが生成されるようにした．次に，Clang コンパイラを用いて変換後の C プログラムを uBPF バイトコードへコンパイルする．追加した外部関数の定義を一緒にコンパイルすることで，P4 プログラムの uBPF バイトコードから外部関数が呼び出されるようにする．また，外部関数の中でのヘルパー関数の呼び出しは，ヘルパー関数に割り当てられた番号をアドレスとする call 命令にコンパイルされる．

4.4 P4 VM とユーザ VM 間の共有メモリ

P4Shield は QEMU の ivshmem 機能を用いて，P4 VM とユーザ VM の間に共有メモリを確立する．まず，ホスト上で ivshmem-server を実行し，共有メモリとして用いるメモリ領域を確保する．そのメモリ領域にアクセスするための仮想的な PCI デバイスを P4 VM とユーザ VM に提供することにより，P4 VM とユーザ VM がメモリを共有できるようにする．ユーザ VM では，OS カーネルがこの仮想 PCI デバイスのメモリ領域に直接アクセスすることで共有メモリを読み書きすることができる．

P4 VM では，ユーザランドで動作する uBPF ランタイムが共有メモリにアクセスできるようにするために，仮想 PCI デバイスを uio デバイスとして提供する．uio はユーザランドからデバイスのメモリへのアクセスを可能にする Linux のインタフェースである．既存の ivshmem-uio ドライバでは read や write などのシステムコールを用いた読み書きしかサポートされていなかったため，先行研究 [18] で開発されたドライバを用いた．このドライバを用いることで，mmap システムコールを用いて uio デバイスをメモリにマップして共有メモリに直接，読み書きを行うことができる．

5. 実験

P4Shield が P4 プログラムを用いてパケットフィルタリングを行えるかどうかを確認する実験を行った．そのために，ユーザ VM 内で情報を共有メモリに格納しておき，P4 VM 内の P4 プログラムがその情報を取得してパケットフィルタリングを行った．また，P4Shield を用いることによって生じるオーバヘッドを調べるために，ユーザ VM の通信性能および P4 プログラムの実行による性能への影響を計測した．実験環境を**表 1**，**表 2** に示す．

5.1 動作確認

P4Shield を用いて不正な ICMP パケットだけを OVS において破棄できることを確認する実験を行った．この実験では，ユーザ VM が UDP パケットを送信していない時に，UDP パケットの不着を示す ICMP パケットを受信す

表 1: 実験環境 (ホスト)

	サーバ	クライアント
CPU	AMD EPYC 7713P	Intel Core i7-12700
メモリ	128 GB	64 GB
OS	Linux 6.5	Linux 6.5
NIC	Broadcom 57416	Intel I219-V
ハイパーバイザ	QEMU-KVM 6.2.0	–
仮想スイッチ	Open vSwitch 3.2.1	–

表 2: 実験環境 (VM)

	P4 VM	ユーザ VM
仮想 CPU	16	8
メモリ	16 GB	8 GB
OS	Linux 5.15	Linux 5.15

ると不正なパケットとみなすようにした．この ICMP のタイプは 3 (Destination Unreachable)，コードは 3 (Port Unreachable) である．

まず，ユーザ VM が UDP パケットを送出したという情報を共有メモリに書き込んでおき，クライアントからユーザ VM に対して 2 種類の ICMP パケットを 3 つずつ送出した．1 つはタイプが 3，コードが 3 の ICMP パケットであり，もう 1 つは ping に用いられる，タイプが 8 (Echo Request) の ICMP パケットとした．その結果，ユーザ VM がこれらの ICMP パケットを正しく受信できていることが確認できた．加えて，ping の返答に用いられる，タイプが 0 (Echo Reply) の ICMP パケットがユーザ VM から送信できていることも確認できた．

続いて，ユーザ VM が UDP パケットを送出していないという情報を共有メモリに書き込んでおき，上記と同じ 2 種類の ICMP パケットをユーザ VM に対して送出した．その結果，タイプが 3，コードが 3 の ICMP パケットは正しく破棄されることが確認できた．その一方で，遮断対象ではないタイプが 8 や 0 の ICMP パケットは正常に転送できていることが確認できた．このことより，P4 Shield は不正な攻撃パケットのみを破棄できることが示された．

5.2 通信性能

Netperf 2.7.0 を用いて，P4 Shield における TCP および UDP のレイテンシとスループットを測定した．UDP については，1472 バイトを超える送信サイズを指定して IP フラグメンテーションが発生すると P4 Shield では正しい測定結果が取得できなかったため，送信サイズを 1472 バイトとした．比較として，P4 に対応していない従来の OVS を用いた場合についても測定を行った．4.1 節で述べたように，OVS は最大で 32 個のパケットを 1 つのバッチにまとめて処理を行っているが，P4 Shield では実装を簡単にするために 1 つのバッチで 1 個のパケットのみを処理している．P4 Shield と同じ条件で測定を行うために，1 バッチに含めるパケット数を 1 とした OVS についても測定を行った．ここでは，このように改変した OVS を OVS(1) と呼び，従来の OVS を OVS(32) と呼ぶ．

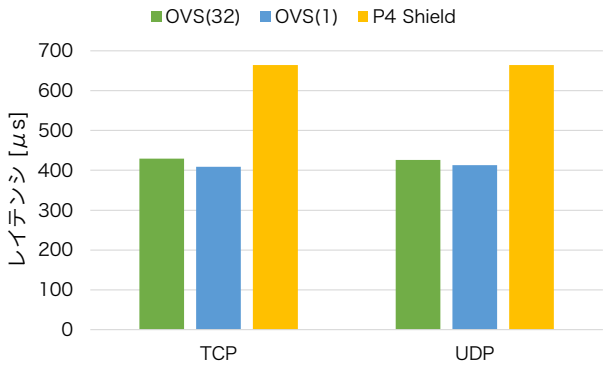

図 3: TCP と UDP のレイテンシ

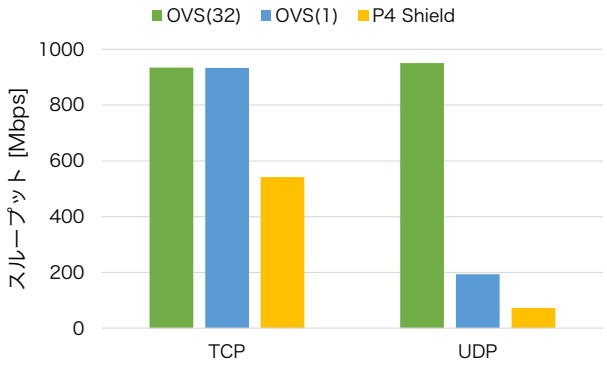

図 4: TCP と UDP のスループット

30 秒間計測したレイテンシの平均値を**図 3** に示す．OVS(32) のレイテンシは OVS(1) より少しだけ大きかったが，これは最大 32 パケットをバッチでまとめて処理したためだと考えられる．OVS(1) と比べると，P4 Shield のレイテンシは TCP でも UDP でも 60 %増加した．これは P4 VM と通信して P4 プログラムを実行したことによるオーバヘッドである．

次に，5 回ずつ測定したスループットの平均値を**図 4** に示す．TCP については，OVS(32) と OVS(1) の性能はほぼ同じであったが，P4 Shield を用いると性能が 42%低下した．一方，UDP については，OVS(32) と比較すると性能が 92 %も低下した．OVS(1) の性能は OVS(32) の 20 %であったため，OVS では複数パケットをバッチにまとめて処理を行うことが UDP のスループットに大きく寄与していることが分かった．OVS(1) と比べた場合には，P4 Shield の性能低下は 62 %となった．このことから，P4 Shield でもパケットをバッチ処理できるように実装することで UDP のスループットを向上させられると考えられる．

送信サイズを 1472 バイトよりも小さくした時の UDP のスループットを**図 5** に示す．OVS(32) では送信サイズを

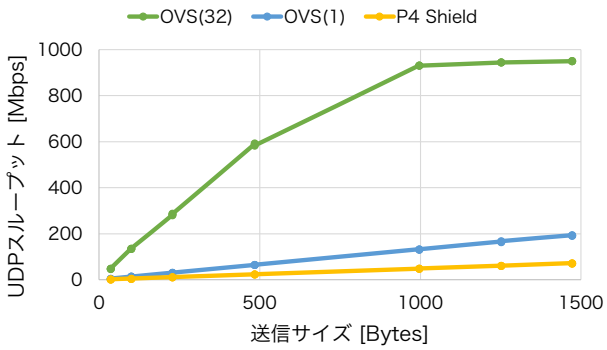

図 5: 送信サイズごとの UDP のスループット

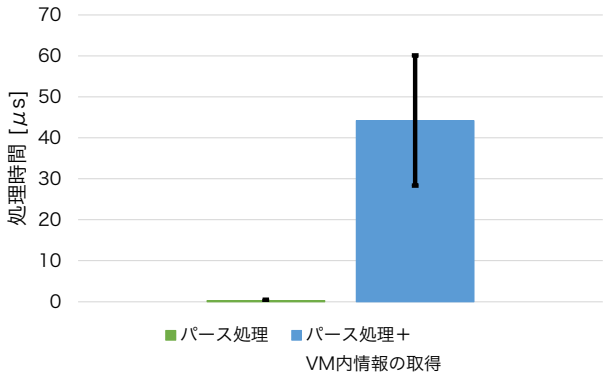

図 6: P4 プログラムの実行時間

1000 バイトにしたあたりでスループットが向上しなくなった．これはネットワーク帯域がボトルネックになったためである．一方，OVS(1) や P4 Shield は送信サイズを増やすと 1472 バイトまでスループットが向上した．

5.3 オーバヘッド

P4 Shield で追加された処理のオーバヘッドを調べるための実験を行った．まず，P4 VM における P4 プログラムの実行時間を調べた．この実験では，(1) P4 プログラムにおいてパケットヘッダのパース処理のみを行った場合と，(2) P4 プログラムにおいてパース処理および VM 内情報の参照を行った場合についてそれぞれ 100 パケット分測定して平均を求めた．P4 プログラムを実行するのにかかった時間を**図 6** に示す．この結果より，P4 プログラムで VM 内情報の取得を行うと大幅に実行時間が長くなり，ばらつきも大きくなることが分かった．

次に，P4 Shield において P4 の処理にかかる時間の内訳を調べた．この実験では，OVS において P4 VM にパケットデータの転送処理を開始する直前から，P4 プログラムの実行結果の受信処理を完了するまでの時間を測定した．また，P4 VM において P4 プログラムの実行に要する時間を計測し，その差から OVS と P4 VM の間の通信時間を算出した．10 回計測した平均値を**図 7** に示す．この結果より，P4 の処理に要する時間の 85 %が OVS と P4 VM の通信に

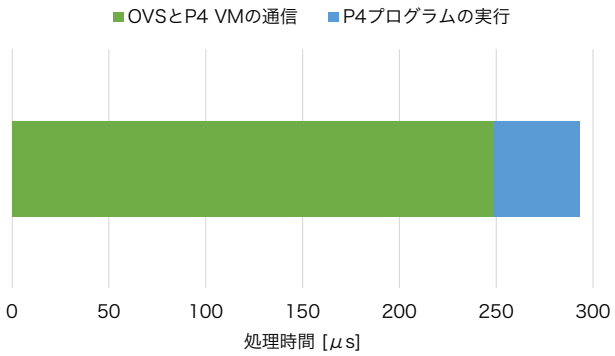

図 7: P4 の処理時間の内訳

費やされていることが分かった．これは P4 Shield が OVS と P4 VM の通信に TCP ソケット通信を用いているためである．共有メモリを使用した通信を行えるようにすることにより，このオーバヘッドは削減できると考えている．

6. 関連研究

P4rt-OVS [15] は OVS 2.13 ベースの仮想 P4 スイッチである．P4 プログラムは ovs-ofctl コマンドを用いて仮想スイッチにロードされ，パケット到着時に仮想スイッチ内で実行される．P4 プログラムは P4Runtime や p4c に組み込まれた p4c-ubpf を用いて uBPF バイトコードにコンパイルされる．P4 Shield と異なり仮想スイッチ内で uBPF ランタイムが動作するが，uBPF により仮想スイッチは P4 プログラムから保護される．ユーザが P4 プログラムをロードすることは想定されておらず，P4 プログラムが VM 内情報を利用することもできない．

EndBox [11] は Intel SGX を用いて，ファイアウォールや侵入検知システムなどのミドルボックスの機能をクライアント側で安全に実行することを可能にしている．クライアント側で動作する EndBox クライアントは SGX エンクレイヴ内に暗号通信のための鍵を保持し，組織やプロバイダ内の EndBox サーバに VPN を用いて接続する．クライアントのアプリケーションは EndBox サーバを経由しない限り通信を行うことができないため，EndBox の利用を強制することができる．

SGX-Box [12] は Intel SGX を用いてミドルボックス内で暗号化トラフィックを安全に復号することを可能にしている．SGX エンクレイヴ内に保持するセッション鍵を用いてパケットを復号し，侵入検知などを行う．SGX-Box は暗号化トラフィックを処理するための SB lang と呼ばれる高レベルプログラミング言語を提供する．SB lang は TLS ハンドシェイクの分析，パケット再構成，復号・再暗号化などの詳細を開発者から隠蔽する．その点で P4 言語と類似している．

VM 内の情報を用いたきめ細かいパケットフィルタリングが提案されている．VMwall [17] は VM イントロスペ

クションを用いて VM のメモリを解析し，パケットヘッダのポート番号を基にそのパケットを送信したプロセスまたは受信するプロセスを見つける．そして，そのプロセス名がホワイトリストになければパケットを破棄する．xFilter [13] は VMwall に似ているが踏み台攻撃を対象としており，パケットを送信したプロセスやユーザの ID を取得して動的にフィルタリングルールを生成して追加することができる．また，フィルタリング性能を向上させるために様々な最適化を行っている．

7. まとめ

本稿では，VM 内情報を利用できるように拡張した P4 プログラムをユーザごとに用意される P4 VM で安全に実行する P4 Shield を提案した．P4 Shield では，仮想スイッチが受信したパケットを対応する P4 VM に転送して P4 プログラムを実行し，実行結果に基づいてパケットの転送処理を行う．P4 Shield は AMD SEV などを用いて P4 VM をクラウドから保護し，P4 プログラムを P4 VM に隔離することで仮想スイッチを保護する．P4 プログラムは uBPF バイトコードにコンパイルし，uBPF ランタイムを用いて実行する．共有メモリに格納されたユーザ VM 内の情報は新たに定義した外部関数経由で uBPF ランタイムのヘルパー関数を用いて取得する．P4 Shield を Open vSwitch と uBPF ランタイムに実装した．実験の結果，P4 Shield は VM 内情報を利用してパケットを転送・破棄できることが確認できた．また，現在の実装上の問題により，P4 Shield における通信性能が大幅に低下することが分かった．

今後の課題は，P4 Shield における通信性能を改善することである．OVS と P4 VM 間の通信に TCP ソケット通信を用いたことにより，パケットデータの転送と実行結果の返却に時間がかかっている．共有メモリを用いて通信を行うようにすることでより高速なパケット処理が可能になると考えられる．UDP の場合には，パケットをバッチにまとめて処理することで高速化されているため，P4 Shield でも同様の実装を行うことで高速化が期待できる．また，ユーザ VM 内の OS が共有メモリに情報をリアルタイムに格納できるようにすることも必要である．今後，OS のどの情報を P4 プログラムから利用できるようにし，どのような API を用意するかを検討する．

謝辞 本研究の一部は，JST，CREST，JPMJCR21M4 の支援を受けたものである．また，本研究成果の一部は，国立研究開発法人情報通信研究機構（NICT）の委託研究（JPJ012368C05501）により得られたものである．

参考文献

[1] BPF Documentation — The Linux Kernel documentation. https://www.kernel.org/doc/html/latest/bpf/.
[2] Open vSwitch. https://www.openvswitch.org/.
[3] osinstom/P4-OvS: Bringing the power of P4 to OvS! https://github.com/osinstom/P4-OvS.
[4] P4 – Language Consortium. https://p4.org/.
[5] uBPF: Main Page. https://iovisor.github.io/ubpf/.
[6] Mihai Valentin Dumitru, Dragos Dumitrescu, and Costin Raiciu. Can we exploit buggy p4 programs? In *Proceedings of the Symposium on SDN Research*, SOSR '20, pp. 62–68, New York, NY, USA, 2020. Association for Computing Machinery.
[7] Clarence Filsfils, Pablo Camarillo, John Leddy, Daniel Voyer, Satoru Matsushima, and Zhenbin Li. Segment Routing over IPv6 (SRv6) Network Programming. RFC 8986, February 2021.
[8] Clarence Filsfils, Darren Dukes, Stefano Previdi, John Leddy, Satoru Matsushima, and Daniel Voyer. IPv6 Segment Routing Header (SRH). RFC 8754, March 2020.
[9] Clarence Filsfils, Stefano Previdi, Les Ginsberg, Bruno Decraene, Stephane Litkowski, and Rob Shakir. Segment Routing Architecture. RFC 8402, July 2018.
[10] Tal Garfinkel, Mendel Rosenblum, et al. A virtual machine introspection based architecture for intrusion detection. In *Ndss*, Vol. 3, pp. 191–206. San Diega, CA, 2003.
[11] David Goltzsche, Signe Rüsch, Manuel Nieke, Sébastien Vaucher, Nico Weichbrodt, Valerio Schiavoni, Pierre-Louis Aublin, Paolo Cosa, Christof Fetzer, Pascal Felber, Peter Pietzuch, and Rüdiger Kapitza. EndBox: Scalable Middlebox Functions Using Client-Side Trusted Execution. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 386–397, June 2018.
[12] Juhyeng Han, Seongmin Kim, Jaehyeong Ha, and Dongsu Han. SGX-Box: Enabling Visibility on Encrypted Traffic using a Secure Middlebox Module. In *Proceedings of the First Asia-Pacific Workshop on Networking*, APNet '17, pp. 99–105, New York, NY, USA, 2017. Association for Computing Machinery.
[13] Kenichi Kourai, Takeshi Azumi, and Shigeru Chiba. Efficient and fine-grained vmm-level packet filtering for self-protection. *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS)*, Vol. 5, No. 2, pp. 83–100, 2014.
[14] Mallik Mahalingam, Dinesh Dutt, Kenneth Duda, Puneet Agarwal, Larry Kreeger, T. Sridhar, Mike Bursell, and Chris Wright. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. RFC 7348, August 2014.
[15] Tomasz Osiński, Halina Tarasiuk, Paul Chaignon, and Mateusz Kossakowski. P4rt-OVS: Programming Protocol-Independent, Runtime Extensions for Open vSwitch with P4. In *2020 IFIP Networking Conference (Networking)*, pp. 413–421, June 2020.
[16] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 117–130, Oakland, CA, May 2015. USENIX Association.
[17] Abhinav Srivastava and Jonathon Giffin. Tamper-Resistant, Application-Aware Blocking of Malicious Network Connections. In Richard Lippmann, Engin Kirda,

and Ari Trachtenberg, editors, *Recent Advances in Intrusion Detection*, pp. 39–58, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[18] 能野智玄, 光来健一. AMD SEV を用いてメモリが暗号化された VM に対する IDS オフロード. 情報処理学会研究報告. [システムソフトウェアとオペレーティング・システム], Vol. 2021-OS-153, No. 1, pp. 1–8, 07 2021.