AF_XDPを用いた安全かつ仮想スイッチ非依存のP4実行基盤

田代 滉太[1]　岩井 正輝[1]　光来 健一[1]

概要：近年，プログラム可能なネットワークスイッチにおいて，P4 言語を用いた柔軟なパケット転送制御が可能となっている．仮想スイッチ上で P4 プログラムを実行できる仮想 P4 スイッチも開発されているが，ユーザが自身の仮想マシン（VM）内の情報を利用する P4 プログラムをロードできればよりきめ細かいパケット制御が可能となる．しかし，クラウドにおいては仮想スイッチからユーザの P4 プログラムへの攻撃や，ユーザの P4 プログラムから仮想スイッチへの攻撃を考慮する必要がある．先行研究では P4 プログラムを機密 VM として動作する P4 VM で実行することにより相互保護を実現しているが，パケットを P4 VM に転送する処理は仮想スイッチに依存している．本稿では，AF_XDP ソケット API を利用する仮想スイッチに対して，安全かつ透過的に P4 プログラムを実行することを可能にする TransP4 を提案する．TransP4 は OS カーネル内でパケットをコピーすることなく P4 VM に転送し，P4 プログラムの実行結果に基づいてパケット転送を制御する．AF_XDP はユーザ空間とデバイスドライバ間で高速なパケット転送を行うが，TransP4 はカーネル内の AF_XDP サブシステムの拡張のみで実現される．TransP4 を Linux に実装し，通信レイテンシは約 50〜60%増加する一方，UDP スループットの低下は 1%未満に抑えられることを確認した．

1. はじめに

近年，ネットワークにおけるトラフィック制御や計測，セキュリティ機能の高度化に伴い，パケットを柔軟に制御できる仕組みが求められている．この解決策として，パケット処理をプログラムとして記述できる P4 [3] が提案されている．ネットワークスイッチにおいて P4 プログラムを実行することにより，データプレーンのパケット処理をソフトウェアで制御することができる．仮想マシン（VM）が接続される仮想ネットワークで用いられている仮想スイッチにおいても，P4 プログラムを実行可能な仮想 P4 スイッチが開発されている [9]．

ユーザが自身の VM 内の情報を利用する P4 プログラムを仮想スイッチにロードすることができれば，システムやアプリケーションの状態に応じたきめ細かなパケット制御が可能となる．しかし，クラウドにおいては仮想スイッチからユーザの P4 プログラムへの攻撃や，ユーザの P4 プログラムから仮想スイッチへの攻撃といったセキュリティ上の脅威を考慮する必要がある．先行研究 [7] では，P4 プログラムを機密 VM として実行される専用 VM（P4 VM）上で実行することで，P4 プログラムと仮想スイッチの相互保護を実現している．しかし，パケットを P4 VM に転送するために仮想スイッチの内部実装を変更する必要があり，特定の仮想スイッチに強く依存する．

そこで本稿では，仮想スイッチに対して安全かつ透過的に P4 プログラムを実行することを可能にする TransP4 を提案する．TransP4 は，VM が送受信するパケットの情報を OS カーネル内で P4 VM に転送することにより，ユーザ空間で動作する仮想スイッチへの変更を不要にする．TransP4 はカーネルのネットワークスタックをバイパスすることを可能にする AF_XDP ソケット API [13] を用いる仮想スイッチを対象とする．AF_XDP はユーザ空間とデバイスドライバ間で高速なパケットのやりとりを実現しているため，デバイスドライバ層において P4 VM と通信して P4 プログラムを実行する．

TransP4 は，既存のデバイスドライバには変更を加えず，カーネル内の AF_XDP サブシステムのみを変更する．一方で，既存の仮想スイッチとの互換性を維持するために，AF_XDP ソケット API は変更しない．仮想スイッチは従来通り，カーネルとの間で共有される UMEM と呼ばれるメモリ領域を用いてパケットデータを扱うことができる．TransP4 では，UMEM を P4 VM とも共有することにより，ゼロコピーでのパケット処理を維持する．また，仮想スイッチは従来通り，TX リングおよび RX リングと呼

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

ばれるリングバッファを用いて AF_XDP サブシステムとディスクリプタのやりとりを行うことができる．TransP4 では，そのディスクリプタを P4 VM とのやりとりにも利用する．

TransP4 を Linux 6.8 に実装し，ユーザ VM 内のネットワーク利用率に関する情報を利用する P4 プログラムを用いて高度なパケットフィルタリングが可能であることを確認した．また，TransP4 を用いた場合のユーザ VM の通信性能を測定し，レイテンシが 50〜60%増加することが分かった．UDP スループットの低下は 1%未満にとどまる一方で，TCP スループットについては大きなばらつきが見られた．P4 VM に処理遅延を挿入するとレイテンシの増加およびスループットの低下が生じるが，TCP スループットは安定することが分かった．これにより，P4 VM の処理性能が通信性能を左右する重要な要因であることが確認できた．

以下，2 章で仮想 P4 スイッチと先行研究の問題点を挙げる．3 章では安全かつ仮想スイッチ非依存の P4 実行基盤である TransP4 を提案する．4 章で TransP4 の実装について説明し，5 章で TransP4 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. 仮想 P4 スイッチ

近年，仮想スイッチにおいて P4 プログラムを実行可能な仮想 P4 スイッチが開発されている．P4 は，ネットワークスイッチのデータプレーンにおけるパケット処理を記述するためのプログラミング言語である．P4 プログラムはパケットヘッダを抽出し，ヘッダ情報に基づくパケット処理を行った後，パケットを再構築する．これにより，パケットの解析や転送処理をソフトウェアとして柔軟に定義し，データプレーンの処理内容をプログラムにより変更することができる．仮想 P4 スイッチは，VM が送受信するパケットに対して P4 プログラムを適用することで，仮想ネットワークにおいて柔軟な転送制御を実現する．

クラウドにおいて，仮想 P4 スイッチへの P4 プログラムのロードはクラウド管理者によって行われることが一般的である．これは仮想ネットワークを管理しているのがクラウド管理者であるためである．クラウドのユーザが P4 プログラムを仮想 P4 スイッチにロードできるようになれば，ユーザは独自の P4 プログラムを作成し，ユーザの VM ごとにより柔軟なパケット転送処理を行うことが可能になる．さらに，P4 プログラムの処理においてパケットのヘッダ情報だけでなく，ユーザの VM 内の情報も利用できれば，VM 内のシステムの状態やアプリケーションの利用状況に応じたより高度な制御が実現できる．

しかし，クラウドの仮想 P4 スイッチとユーザの P4 プログラムは互いに信頼できるとは限らない．仮想 P4 スイッチはユーザによってロードされた P4 プログラムを盗聴することにより，P4 プログラム内部に格納された情報や，P4 プログラムが取得した VM 内の情報を不正に入手できる可能性がある．また，これらの情報を改ざんされるとユーザが意図しないパケット処理を行われる恐れがある．逆に，ユーザの P4 プログラムの不正な動作によって仮想 P4 スイッチに影響を及ぼす可能性もある．例えば，過剰な計算処理やループ処理を含む P4 プログラムがロードされた場合，パケット処理の遅延やスループット低下を引き起こし，仮想 P4 スイッチの計算資源を枯渇させる可能性がある [5]．このような影響は仮想 P4 スイッチ経由で他の VM の通信にも波及する恐れがある．

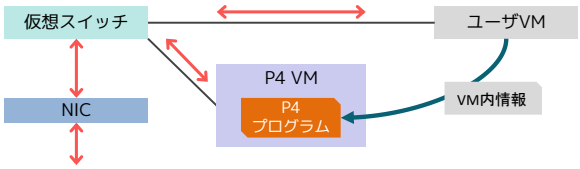

図 1 P4 Shield のシステム構成

そこで，先行研究の P4 Shield [7] では，図 1 のように P4 プログラムをユーザごとに用意した専用の VM（P4 VM）上で隔離実行し，クラウドにおいて安全にユーザの P4 プログラムを実行する．P4 Shield は，クラウドの仮想スイッチが受信したパケットを一旦，P4 VM に転送し，P4 VM 内で P4 プログラムを安全に実行した後，その結果に基づいて仮想スイッチがパケットの転送処理を行う．P4 Shield は P4 VM を機密 VM として実行することで，P4 プログラムおよび VM 内情報をクラウドから保護する．機密 VM はハードウェアによって提供される高信頼実行環境（TEE）を用いて保護された VM である．それに加えて，P4 VM 内で uBPF [2] を用いて P4 プログラムを実行することで，VM と uBPF サンドボックスによる隔離により，P4 プログラムがクラウドに及ぼす影響を抑える．

しかし，P4 Shield ではパケットを P4 VM に転送し，P4 プログラムの実行結果を受け取って利用するために，仮想スイッチの内部実装を変更する必要がある．そのため，仮想スイッチの一つである Open vSwitch [1] に強く依存した設計・実装となっており，他の仮想スイッチに適用するには一から設計・実装を行う必要がある．仮想スイッチは大量のパケットを高速に処理する必要があるため，DPDK [6] を用いたり，バッチ処理を行ったりして複雑な実装となっており，P4 Shield を適用した上で高い性能を維持するには細かなチューニングが必要になる．このことから，P4 Shield は新たな仮想スイッチへの適用コストが高く，既存の仮想スイッチの保守性の観点からも課題がある．

3. TransP4

本稿では，仮想スイッチに対して透過的に P4 プログ

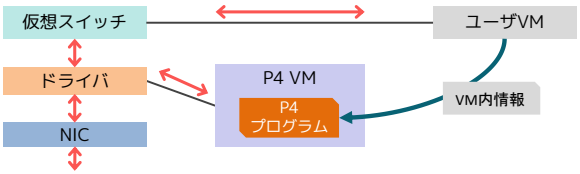

図 2 TransP4 のシステム構成

ラムを実行することを可能にする TransP4 を提案する．TransP4 のシステム構成を図 2 に示す．TransP4 は，送受信するパケットの情報を OS カーネル内で P4 VM に転送して P4 プログラムの実行結果を利用することにより，ユーザ空間で動作する仮想スイッチへの変更を不要にする．これにより，仮想スイッチに依存せずに P4 に対応させることができるようになり，新たな仮想スイッチへの適用や既存の仮想スイッチの保守が容易になる．カーネル内でパケットを扱えるようにするために，カーネルをバイパスする DPDK ではなく，AF_XDP ソケット API [13] を用いる仮想スイッチを対象とする．AF_XDP はカーネルのネットワークスタックをバイパスし，XDP を用いてユーザ空間とデバイスドライバ間で高速なパケットのやり取りを実現する機構である．TransP4 はデバイスドライバ層においてパケットの情報を P4 VM に転送する．

AF_XDP ソケット API においては，パケットデータは UMEM と呼ばれるユーザ空間のメモリ領域に格納され，TX リングおよび RX リングと呼ばれるリングバッファを介してユーザ空間とデバイスドライバ間でのやり取りを行う．仮想スイッチがパケットを送信する際には，パケットデータを UMEM に格納し，そのデータを指すディスクリプタを TX リングに書き込む．その後，デバイスドライバが TX リングからディスクリプタを取り出し，そのディスクリプタが指す UMEM 上のパケットデータを NIC が DMA を用いて送信する．一方，仮想スイッチがパケットを受信する際には，RX リングからディスクリプタを取り出し，そのディスクリプタが指す UMEM 上のパケットデータを取得する．このパケットデータは DMA を用いて NIC によって UMEM に書き込まれ，ディスクリプタはデバイスドライバによって RX リングに書き込まれる．仮想スイッチはこのようにして，パケットデータをコピーすることなく，パケットの送受信を行うことができる．

TransP4 は AF_XDP ソケット API を変更しないようにすることで，既存の仮想スイッチとの互換性を維持する．仮想スイッチは従来通り，パケットデータを UMEM に格納し，そのデータを指すディスクリプタを TX リングに書き込むことでパケットを送信することができる．デバイスドライバはこのパケットを送信する前に，TX リングから取り出したディスクリプタに基づいて P4 VM にパケット情報を転送する．P4 VM は UMEM 上のパケットデータを用いて P4 プログラムを実行し，実行結果をデバイスドライバに返送する．P4 プログラムによってパケットの送信が許可された場合には，NIC が DMA を用いて UMEM 上のパケットデータを送信する．パケットの送信が拒否された場合には，UMEM 上のパケットデータを削除してパケットを破棄する．

同様に，仮想スイッチは従来通り，RX リングからディスクリプタを取り出し，そのディスクリプタが指す UMEM 上のパケットデータを取得することでパケットを受信することができる．UMEM には，NIC がパケットを受信した時に DMA を用いてパケットデータが書き込まれる．その際に，デバイスドライバは RX リングにディスクリプタを書き込む代わりに，P4 VM にパケット情報を転送する．P4 VM は UMEM 上のパケットデータを用いて P4 プログラムを実行し，実行結果をデバイスドライバに返送する．パケットの受信が許可された場合には，デバイスドライバが RX リングにディスクリプタを書き込む．パケットの受信が拒否された場合には，UMEM 上のパケットデータを削除してパケットを破棄する．このようにして，パケット情報を P4 VM に転送してもゼロコピーでパケットの送受信を行うことができる．

P4 VM では P4 ランタイムが動作し，ユーザがロードした P4 プログラムを実行する．P4 プログラムは uBPF バイトコードにコンパイルされ，uBPF サンドボックス内で安全に実行される．P4 プログラムはユーザ VM 内の情報を取得して，パケットの転送可否を判定する．ただし，P4 プログラムは VM 内情報に直接アクセスすることができないため，P4 の外部関数を用いて VM 内情報を取得する．

4. 実装

TransP4 を，デバイスドライバには変更を加えず，Linux カーネル内の AF_XDP サブシステムのみを変更することで実装した．

4.1 AF_XDP

AF_XDP ソケットは，パケットデータを格納する UMEM と，メタデータをやり取りするための 4 つのリングバッファで構成される．UMEM はプロセスがユーザ空間に確保した，ページ境界にアラインメントされたメモリであり，複数のフレームで構成される．setsockopt システムコールを用いて AF_XDP サブシステムと共有され，スワップアウトされないようにカーネル内でピン留めされる．リングバッファのメモリは AF_XDP サブシステムがカーネル内に確保し，プロセスは mmap システムコールを用いてメモリにマップして利用する．

TX リングと RX リングはそれぞれパケットの送信と受信のために用いられるリングバッファである．これらのリングバッファには，パケットデータが格納される UMEM 上のフレームのアドレスおよび長さからなるディスクリプ

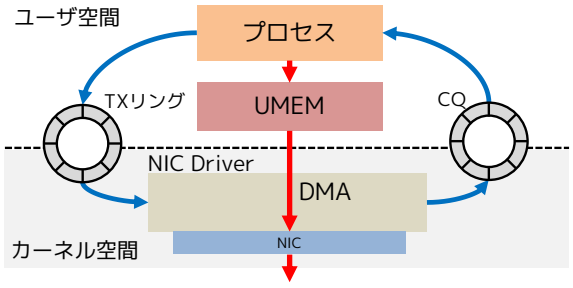

図 3 AF_XDP における送信処理

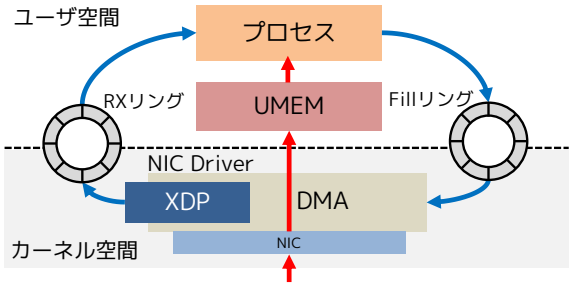

図 4 AF_XDP における受信処理

タが格納される．一方，Fill リングと Completion Queue（CQ）は UMEM フレームの管理に用いられるリングバッファである．Fill リングはプロセスから AF_XDP サブシステムに UMEM の空きフレームを供給するために用いられる．CQ は AF_XDP サブシステムからプロセスに UMEM フレームを返却し，再利用できるようにするために用いられる．これらのリングバッファには UMEM フレームのアドレスのみが格納される．

AF_XDP ソケットを用いた送信処理の流れを図 3 に示す．赤色の矢印はパケットデータの流れを，青色の矢印はディスクリプタの流れを示している．プロセスは送信するパケットデータを UMEM に格納し，そのデータを指すディスクリプタを TX リングに書き込む．そして，sendto システムコールを実行してカーネルにパケット送信を通知すると，カーネル内の AF_XDP サブシステムがデバイスドライバを呼び出す．デバイスドライバは TX リングからディスクリプタを取得し，ディスクリプタが指す UMEM 上のパケットデータを NIC に DMA を用いて送信させる．このとき，パケットデータのコピーは行われず，UMEM 上のデータが直接 NIC によって参照される．送信が完了したパケットのデータについては，デバイスドライバが UMEM フレームのアドレスを CQ に書き込み，対応する UMEM フレームが再利用可能であることをプロセスに通知する．プロセスは CQ からアドレスを取得すると，その UMEM フレームを再利用する．

AF_XDP ソケットを用いた受信処理の流れを図 4 に示す．NIC が受信したパケットは，まずデバイスドライバ内で実行される XDP プログラムによって処理される．XDP プログラムは，受信したパケットを通常のネットワークスタックへ渡すか，AF_XDP ソケットに転送するかを判定する．AF_XDP ソケットに転送される場合，パケットデータは DMA により UMEM 上の空きフレームに直接格納される．受信に使用する UMEM フレームは，プロセスから Fill リングを通じて AF_XDP サブシステムに供給される．AF_XDP サブシステムは Fill リングに登録されたフレームを参照し，受信パケットの DMA 転送先として利用する．その後，デバイスドライバは対応するディスクリプタを RX リングに書き込み，プロセスに通知する．プロセスは RX リングをポーリングすることでディスクリプタを取得し，ディスクリプタが指す UMEM 上のパケットデータを読み込む．

4.2 AF_XDP サブシステムの拡張

TransP4 は，AF_XDP ソケット API を維持したまま，パケット送受信時に非同期に P4 VM にパケットを転送する．従来の AF_XDP サブシステムと同様に，パケットデータのコピーを行わず，リングバッファを用いてディスクリプタのみをやり取りする．

4.2.1 P4 VM との UMEM の共有

AF_XDP サブシステムが P4 VM にパケットデータをコピーせずに転送するには，UMEM を P4 VM とも共有する必要がある．しかし，プロセスのメモリ上に確保された UMEM を VM と共有するのは容易ではない．プロセスと VM がメモリを共有するには，共有メモリ機構を用いる必要があるためである．P4 VM とパケットデータを共有できるようにするために，仮想スイッチが共有メモリ領域を確保し，その領域を UMEM として利用する．仮想スイッチは共有メモリを自身のメモリにマップし，その領域を AF_XDP サブシステムと共有することで，通常の UMEM と同様にパケットの送受信に利用することができる．さらに，P4 VM においても同一の共有メモリ領域にアクセスするために提供される共有メモリデバイスをメモリにマップすることで，UMEM 上のパケットデータを直接参照することができる．この実装では仮想スイッチへの変更が必要になるが，プロセスが確保したメモリをカーネル内で共有メモリに置き換えることで，透過的に共有メモリを UMEM として使用できるようにすることを検討している．

4.2.2 P4-TX リングと P4-RX リング

AF_XDP サブシステムと P4 VM の間でディスクリプタをやり取りするために，P4-TX リングおよび P4-RX リングを用いる．これらのリングは従来の TX リングおよび RX リングと同様に，ディスクリプタのみを格納する．P4-TX リングは，P4 VM での P4 プログラムの実行結果を AF_XDP サブシステムに返送するために用いられる．ディスクリプタには UMEM フレームの情報に加えて，パケットの送受信の方向と P4 プログラムの実行結果が格納

される．P4-RX リングは，P4 プログラムの実行が必要なパケットの情報を P4 VM に転送するために用いられる．ディスクリプタには UMEM フレームの情報とパケットの送受信の方向が格納される．

4.2.3 コンテキストの保存・復元

TransP4 はパケット送受信の処理中に P4 プログラムを非同期に実行するため，実行終了後に送受信処理を再開できるように，必要なコンテキストの保存・復元を行う．そのために，送信処理用に TX-inflight テーブルを用意し，受信処理用に RX-inflight テーブルを用意する．TX-inflight テーブルには処理中の UMEM フレームと AF_XDP ソケットの組を登録しておき，ディスクリプタに格納されている UMEM フレームの情報から AF_XDP ソケットを取り出して送信処理を再開する．一方，RX-inflight テーブルには処理中の UMEM フレームと AF_XDP ソケットバッファの組を登録しておき，UMEM フレームの情報から AF_XDP ソケットバッファを取り出して受信処理を再開する．

4.2.4 ドレインスレッド

P4 プログラムの実行後に AF_XDP サブシステムがディスクリプタを非同期に受け取れるようにするため，ドレインスレッドと呼ばれるカーネルスレッドを用いる．ドレインスレッドは P4-TX リングを定期的に監視し，P4 プログラムから返送されたディスクリプタを取得する．この処理はパケットの送受信処理とは独立して実行されるため，P4 プログラムの実行中にパケット処理がブロックされることを防ぐことができる．ドレインスレッドは取得したディスクリプタに基づいて，パケットの送受信処理を再開する．

4.3 TransP4 におけるパケット送受信

拡張した AF_XDP サブシステムを用いたパケット処理の流れについて説明する．

4.3.1 送信処理

TransP4 における送信処理の流れを図 5 に示す．仮想スイッチは従来の AF_XDP ソケット API を用いて UMEM にパケットデータを格納し，TX リングにそのデータを指すディスクリプタを書き込む．従来の AF_XDP とは異なり，デバイスドライバは TX リングからディスクリプタを取得した後，パケットをすぐには送信しない．その代わりに，AF_XDP サブシステムを呼び出してディスクリプタに送信パケットであることを示す情報を付与し，P4-RX リングに書き込む．同時に，ディスクリプタが指す UMEM フレームと AF_XDP ソケットの組を TX-inflight テーブルに登録する．

P4 VM では，ポーリングを行うことによって P4-RX リングからディスクリプタを取得し，ディスクリプタが指す UMEM 上のパケットデータを参照して P4 プログラムの実行を行う．UMEM は共有メモリ上に確保されているため，P4 VM においてもホストの仮想スイッチやカーネル

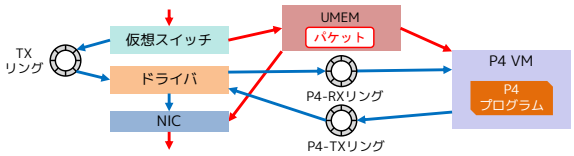

図 5 TransP4 における送信処理

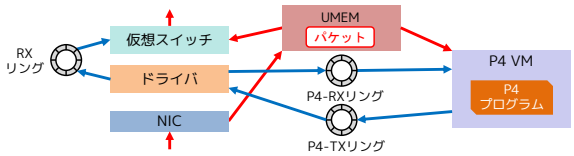

図 6 TransP4 における受信処理

と同様にアクセスすることができる．そして，P4 プログラムの実行結果をディスクリプタに格納し，P4-TX リングに書き込む．

ドレインスレッドは P4-TX リングからディスクリプタを取得すると，送信が許可されたパケットについてはディスクリプタをパスキューと呼ばれる新たなキューに格納する．そして，UMEM フレームを基に TX-inflight テーブルから AF_XDP ソケットを取り出し，それを用いて再度，送信処理を呼び出す．送信処理において，パスキューにディスクリプタが格納されている場合には，UMEM 上のパケットデータを NIC を用いて送信する．送信が完了すると，UMEM フレームは CQ に格納され，仮想スイッチに返却されて再利用が可能となる．一方，送信が拒否されたパケットについては送信処理を行わず，UMEM フレームを CQ に格納してすぐに再利用できるようにする．

4.3.2 受信処理

TransP4 における受信処理の流れを図 6 に示す．NIC から受信したパケットデータは UMEM フレームに格納される．従来の AF_XDP とは異なり，AF_XDP サブシステムはディスクリプタを RX リングに書き込まず，ディスクリプタに受信パケットであることを示す情報を付与し，P4-RX リングに書き込む．同時に，UMEM フレームと AF_XDP ソケットバッファの組を RX-inflight テーブルに登録する．P4 VM では，送信の場合と同様に，UMEM 上のパケットデータを参照して P4 プログラムを実行し，実行結果を格納したディスクリプタを P4-TX リングに書き込む．

ドレインスレッドは P4-TX リングからディスクリプタを取得し，受信が許可されたパケットについてはディスクリプタを RX リングに書き込む．仮想スイッチはポーリングによって RX リングからディスクリプタを取得し，ディスクリプタが指す UMEM フレームのパケットデータを取得する．受信の可否に関わらず，UMEM フレームを基に RX-inflight テーブルから AF_XDP ソケットバッファを取得し，その解放処理を行う．

4.4 P4 ランタイム

P4 VM 内では P4 ランタイムが動作し，P4-RX リングをポーリングすることにより，AF_XDP サブシステムによってディスクリプタが書き込まれるのを待つ．ディスクリプタを取得すると，ディスクリプタが指す UMEM 上のパケットデータを指定して P4 プログラムを実行する．P4 プログラムは uBPF バイトコードにコンパイルされ，ロード時に JIT コンパイルを行うことにより，ネイティブコードとして高速に実行される．その際に uBPF バイトコードの検証も行われるため，uBPF サンドボックス内でユーザの P4 プログラムを安全に実行することができる．P4 プログラムの実行の結果，パケット転送の可否が返される．P4 ランタイムはこの情報をディスクリプタに格納し，P4-TX リングに書き込むことで AF_XDP サブシステムに返送する．

P4 プログラムはユーザ VM 内の情報にアクセスすることができないため，先行研究 [7] と同様に，P4 の外部関数を用いて機能の拡張を行う．外部関数は C 言語で記述され，P4 プログラム内で `extern` を用いて宣言することにより，P4 の通常の関数と同様に呼び出すことができる．定義した外部関数は，ユーザ VM との間の共有メモリ上に格納される VM 内情報を取得して P4 プログラムに返す．しかし，外部関数も uBPF バイトコードにコンパイルして実行されるため，uBPF サンドボックスの制限により，共有メモリに直接アクセスすることはできない．そこで，外部関数は P4 ランタイムが提供するヘルパー関数を呼び出し，P4 ランタイム経由で共有メモリ上の情報にアクセスする．

5. 実験

TransP4 の有効性を確認するために実験を行った．まず，TransP4 を用いて，P4 VM 上で実行される P4 プログラムがユーザ VM 内情報を用いて高度なパケットフィルタリングを実現可能であるかどうかを確認した．次に，TransP4 を適用した場合のユーザ VM の通信性能を測定し，従来システムのおける通信性能との比較を行った．さらに，P4 プログラムの実行時間が通信性能に与える影響を調べた．

本実験では，**表 1** に示す 2 台のホストを使用し，それらを 10GbE スイッチを介して接続した．サーバ上で**表 2** に示す P4 VM とユーザ VM を動作させた．これらの VM は機密 VM として実行しておらず，ユーザ VM 内情報を格納する共有メモリの暗号化も行っていない．サーバ上で動作する仮想スイッチとして，UMEM を P4 VM と共有可能とするよう改変した Open vSwitch 3.5.0 を動作させた．

表 1 実験に用いたホスト

項目	サーバ	クライアント
CPU	Intel Core i7-14700	Intel Core i7-12700
メモリ	32 GB	64 GB
NIC	Intel X540-AT2	
OS	Linux 6.8	Linux 6.11
ハイパーバイザ	QEMU-KVM 6.2.0	–

表 2 実験に用いた VM

項目	P4 VM	ユーザ VM
仮想 CPU	6	2
メモリ	4 GB	4 GB
OS	Linux 5.15	

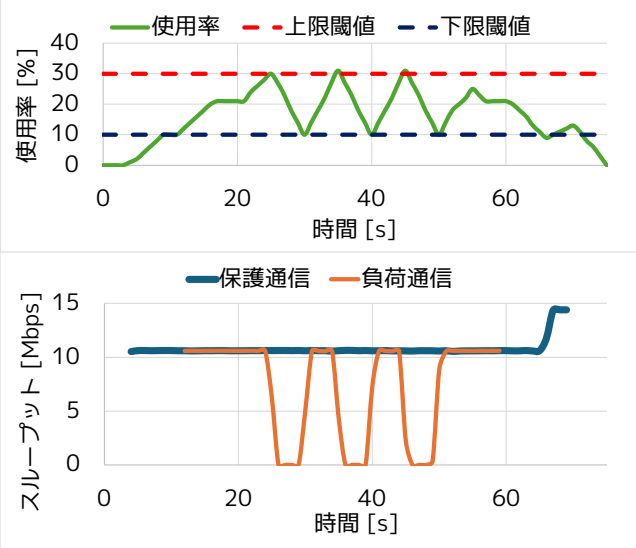

図 7 VM 内情報を用いたパケット制御

5.1 VM 内情報に基づくパケットフィルタリング

P4 プログラムを用いて，高負荷時に負荷の原因となっている通信のみを遮断するパケットフィルタリングを行った．本実験では，8000 番ポートへの UDP 通信（保護通信）のパケットは常に転送し，9000 番ポートへの UDP 通信（負荷通信）を遮断の対象とした．P4 プログラムは共有メモリを介して，ユーザ VM 内で算出されたネットワーク使用率と閾値を取得した．パケット制御にはヒステリシスを用い，ネットワーク使用率が上限閾値（30%）を超えた場合に負荷通信の遮断を開始し，下限閾値（10%）を下回った場合に遮断を解除する．ネットワーク使用率は UDP 通信における受信パケット数から算出した．

図 7 に実験結果を示す．上段のグラフはネットワーク使用率と閾値の関係を示しており，ネットワーク使用率が上限閾値を超えたタイミングで負荷通信が遮断されてネットワーク使用率が低下し，下限閾値を下回ると遮断が解除されてネットワーク使用率が上昇した．その結果，下段のグラフに示すように負荷通信のスループットは周期的に低下する一方で，保護通信のスループットはほぼ一定に維持された．

5.2 通信性能

netperf 2.7.0 [8] を用いてクライアントからユーザ VM へパケットを送信し，TransP4 における UDP および TCP

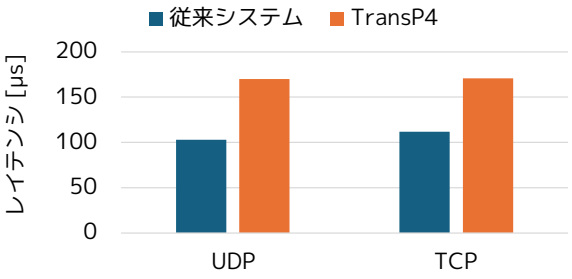

図 8 レイテンシ

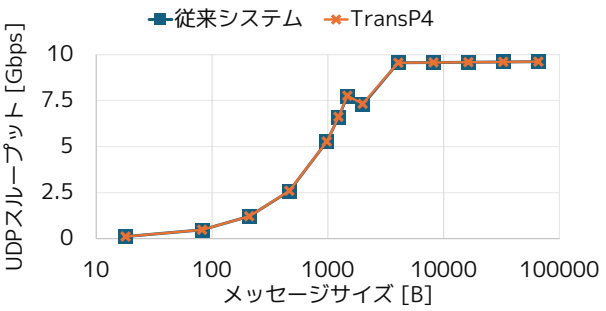

図 9 UDP スループット

のレイテンシとスループットを測定した．この実験では，すべてのパケットの転送を許可する P4 プログラムを用いた．比較として，AF_XDP ソケット API を用いる従来システムについても同様の測定を行った．

5.2.1 レイテンシ

TransP4 が通信レイテンシに与える影響を調べた．レイテンシの測定には，netperf の omni テストの Request/Response（RR）を用いた．UDP と TCP のそれぞれについて，10 回ずつ測定した時の P90 レイテンシの平均を図 8 に示す．TransP4 を適用した場合，UDP と TCP のいずれにおいても約 50〜60% のレイテンシ増加が確認された．これは，パケットを P4 VM へ転送し，P4 プログラムによる判定処理を行うオーバヘッドが追加されたためである．

5.2.2 スループット

TransP4 が通信スループットに与える影響を調べた．スループットの測定には，netperf の UDP_STREAM および TCP_STREAM テストを使用した．一度に送信するメッセージサイズを変えながら 10 回ずつ測定し，平均スループットを算出した．

図 9 に UDP スループットを示す．UDP 通信においては，メッセージサイズを増やすとスループットが向上し，従来システムと TransP4 のいずれにおいても約 9.6 Gbps で飽和した．また，どのメッセージサイズにおいても TransP4 によるスループット低下は 1%未満にとどまっており，スループットへの影響は限定的であることが確認された．

図 10 に TCP スループットを示す．TransP4 では測定値に大きなばらつきが見られたため，外れ値を含めた場合と

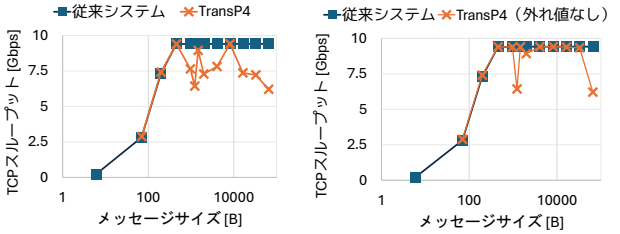

図 10 TCP スループット

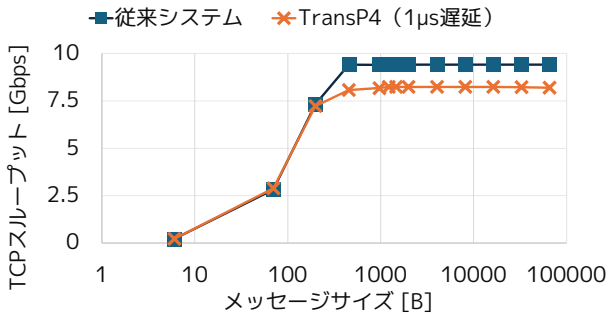

図 11 TCP スループット（1 μs の処理遅延）

除去した場合の両方の結果を示す．従来システムでは，すべてのメッセージサイズにおいて安定したスループットが得られており，約 9.4 Gbps で飽和した．一方で，TransP4 ではメッセージサイズが 6 バイトの時には通信が安定せず，スループットの測定が行えなかった．また，大きなメッセージサイズにおいてはスループットに大きなばらつきが観測された．外れ値を除去した場合には従来システムと同様に約 9.4 Gbps で飽和することが確認された．しかし，一部のメッセージサイズにおいては測定値の分布が広く，外れ値として除去されない測定値が平均に影響を与えている．

このばらつきの原因を調査した結果，同一の UMEM フレームが再利用される前に複数回，CQ に格納されていることが確認された．その結果，再利用可能なフレーム数が想定以上に増加して仮想スイッチにおいてエラーが発生し，NIC のリセットが引き起こされていることが分かった．その際に通信が一時的に停止し，スループットの低下として観測された．このように，UMEM フレームの管理に問題があることが分かった．

このような現象が発生する条件を調べるために，P4 VM に 1 μs の処理遅延を挿入して TCP スループットを測定した．図 11 に示すように，この条件ではすべてのメッセージサイズにおいてスループットが安定し，ばらつきはほとんど見られなくなった．このことから，P4 VM による判定処理のタイミングが UMEM フレームの管理に影響を与えている可能性があることが分かった．

5.3 P4 プログラムの実行時間の影響

P4 VM 上で実行される P4 プログラムの実行時間が通信

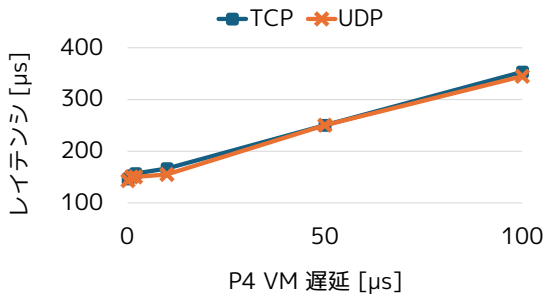

図 12 レイテンシへの P4 VM の処理遅延の影響

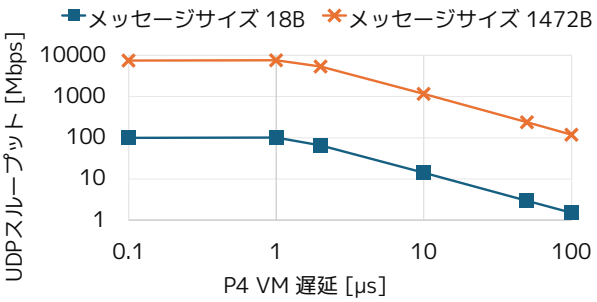

図 13 UDP スループットへの P4 VM の処理遅延の影響

性能に与える影響を調査するために，P4 VM に処理遅延を挿入してレイテンシおよび UDP スループットを測定した．処理遅延として，P4 プログラムの実行後に 0 μs から 100 μs のビジーウェイトを行った．

図 12 に P4 VM の処理遅延とレイテンシの関係を示す．測定値は往復レイテンシであるため，パケットの送信時と受信時に挿入される 2 回の処理遅延の影響が含まれている．処理遅延が小さい場合にはレイテンシへの影響は限定的であるが，遅延が大きくなるとそれに比例してレイテンシも増加することが確認された．

図 13 に P4 VM の処理遅延と UDP スループットの関係を示す．処理遅延が小さい場合にはスループットへの影響は小さいが，遅延が増加するにつれてスループットが大幅に低下することが確認された．また，処理遅延の増加に伴うスループット低下の割合は，メッセージサイズによらずほぼ同様であることが分かる．これは，P4 VM を用いた処理がパケットサイズに依存しない固定的なオーバヘッドになっているためである．

6. 関連研究

P4rt-OVS [9] は Open vSwitch をベースとした仮想 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部で実行される．P4 プログラムは uBPF バイトコードにコンパイルされ，仮想スイッチ内の uBPF ランタイム上で実行される．uBPF により一定の安全性は確保されるが，uBPF ランタイムは仮想スイッチに統合されており，スイッチの実装に大きく依存する．また，管理者による P4 プログラムのロードを前提としており，ユーザによる柔軟な利用は想定されていない．

P4 Shield [7] はクラウドにおいてユーザの P4 プログラムを安全に実行するためのシステムである．仮想スイッチからユーザごとに用意される専用 VM（P4 VM）にパケットを転送し，P4 プログラムを実行する．P4 VM を機密 VM として実行することで，P4 プログラムおよび P4 プログラムが利用する VM 内情報をクラウドから保護する．また，VM および uBPF サンドボックスによる隔離により，P4 プログラムがクラウドに及ぼす影響を抑える．一方で，パケット転送のために仮想スイッチの内部実装を変更する必要があり，特定の仮想スイッチに依存する．

DPDK [6] はユーザ空間から NIC を直接操作することにより，低遅延かつ高スループットなパケット処理を実現するためのフレームワークである．AF_XDP とは異なり，カーネルを完全にバイパスするため，ユーザ空間で NIC のデバイスドライバを実行する．ユーザ空間で動作する仮想スイッチにも利用されており，Open vSwitch にもデータパスの一つとして実装されている．パケット処理がユーザ空間で完結するため，カーネル内で P4 VM にパケットを転送することはできないが，ユーザ空間の DPDK ライブラリを改変して P4 VM にパケットを転送することはできると考えられる．

netmap [11] はカーネルのネットワークスタックをバイパスし，高速なパケット入出力を実現するための機構である．netmap は AF_XDP によく似ており，カーネル内のパケットバッファやリングバッファをプロセスと共有してパケットの送受信を行う．VALE [12] と呼ばれるカーネル内の仮想スイッチを併用することにより，高速にパケット転送を行うことができる．VALE を拡張して P4 VM にパケットを転送することも可能であると考えられる．

FLASH [4] は同一ホスト上で動作しているネットワークファンクション間で，AF_XDP ソケットを用いてパケット転送をゼロコピーで行うことを可能にする．そのために，AF_XDP ソケット間で UMEM を共有し，リングバッファへの同時書き込みと同時読み込みを可能にしている．TransP4 は仮想スイッチと P4 VM の間でパケット転送をゼロコピーで行うことを可能にしており，VM も扱っている点が FLASH とは異なる．P4 VM 内では AF_XDP ソケットは用いていないが，AF_XDP ソケット API とほぼ同じ UMEM と P4-TX リング，P4-RX リングを扱っている．

Toasty [10] は AF_XDP において再利用可能な UMEM フレームをキューではなくスタックで管理することにより，CPU キャッシュのヒット率を向上させる．さらに，デバイスドライバが NIC の RX リングに入れるディスクリプタ数を制御することで，使用される UMEM フレーム数を抑え，キャッシュにヒットしやすくする．TransP4 では

UMEM は P4 VM からもアクセスされるため，キャッシュヒット率を高めるにはさらに複雑な制御が必要になる．

7. まとめ

本稿では，AF_XDP ソケット API を用いる仮想スイッチに対して，安全かつ透過的に P4 によるパケット転送制御を実現する TransP4 を提案した．TransP4 は仮想スイッチが用いる API を変更することなく，OS カーネル内の AF_XDP サブシステムにおいてパケットをコピーせずに P4 VM に転送する．P4 VM では P4 プログラムがユーザの VM 内情報を利用してパケットの転送可否を判定し，AF_XDP サブシステムがパケットを転送または破棄する．実験により，ユーザ VM 内情報に基づいてパケット転送を動的に制御できることを確認した．また，レイテンシは 50〜60%増加するが，UDP スループットの低下は 1%未満にとどまることが分かった．一方，TCP スループットについては大きなばらつきが観測された．

今後の課題として，負荷の高い TCP 通信を安定して行えるようにすることが挙げられる．UMEM フレーム管理の問題を解消して NIC がリセットされないようにする必要がある．また，P4 VM として機密 VM を用い，共有メモリを保護した場合の性能についても測定する予定である．

謝辞 本研究の一部は，JST，CREST，JPMJCR21M4 の支援を受けたものである．

参考文献

[1] Open vSwitch. `https://www.openvswitch.org/`.
[2] uBPF: Main Page. `https://iovisor.github.io/ubpf/`.
[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, Vol. 44, No. 3, pp. 87–95, 2014.
[4] D. Das, K. P. Baua, A. Kansara, A. Chakraborty, D. Kurukunda, M. Vutukuru, and P. Kulkarni. FLASH: Fast Linked AF_XDP Sockets for High Performance Network Function Chains. In *Proceedings of the 2025 ACM Symposium on Cloud Computing*, pp. 571–584, 2026.
[5] M. V. Dumitru, D. Dumitrescu, and C. Raiciu. Can We Exploit Buggy P4 Programs? In *Proceedings of the Symposium on SDN Research*, pp. 62–68, 2020.
[6] Intel Corp. DPDK – The Open Source Data Plane Development Kit. `https://www.dpdk.org/`.
[7] M. Iwai and K. Korai. P4 Shield: Secure Execution of Users' P4 Programs with In-VM Information inside Clouds. In *IEEE Computing and Communication Workshop and Conference*, 2026.
[8] R. Jones. Netperf Benchmark. `https://hewlettpackard.github.io/netperf`.
[9] T. Osiński, H. Tarasiuk, P. Chaignon, and M. Kossakowski. P4rt-OVS: Programming Protocol-Independent, Runtime Extensions for Open vSwitch with P4. In *2020 IFIP Networking Conference (Networking)*, pp. 413–421, 2020.
[10] Preeti, N. Bhat, A. Kumar, and M. Vutukuru. Toasty: Speeding Up Network I/O with Cache-Warm Buffers. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 2015–2029, 2026.
[11] L. Rizzo. netmap: A Novel Framework for Fast Packet I/O. In *Proceedings of the USENIX Annual Technical Conference*, 2012.
[12] L. Rizzo and G. Lettieri. VALE, a Switched Ethernet for Virtual Machines. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, pp. 61–72, 2012.
[13] B. Töpel and M. Karlsson. AF_XDP: High Performance Networking in Linux. In *Linux Plumbers Conference*, 2018.