Vol.2014-IOT-26 No.10
2014/6/28

IPsecトンネリングにおけるパケットフロー単位での並列処理手法の提案

小川 拡[1,a)]　齋藤 彰一[1]　川島 龍太[1]　瀧本 栄二[2]　毛利 公一[2]　松尾 啓志[1]

概要：マルチキュー NIC の登場により，パケットの並列処理が実現されている．しかし，IPsec を利用してルータ間でトンネリングする場合，パケットは転送元のルータでヘッダを含めて暗号化されるため，転送先のルータの NIC ですべてのパケットが同一フローとして扱われ，割り込みが CPU の特定の 1 コアに集中して性能を低下させる．本稿ではこの状況に対して，トンネルの両端のルータが連携する IPsec パケットの並列処理手法を提案する．転送元のルータで暗号化前にフロー識別情報を生成して暗号化後のパケットに付与することで，転送先のルータの NIC では情報に基づいてフローを識別し，フロー単位で割り込みの発生先を切り替えることでトンネリングの高速化を実現する．本稿では，フロー識別情報として IP アドレスを用いた手法について述べる．

キーワード：IPsec，トンネリング，マルチキュー NIC，並列処理，パケットフロー

A Proposal for Parallelization of IPsec Tunneling by Packet Flows

Hiromu Ogawa[1,a)]　Shoichi Saito[1]　Ryota Kawashima[1]　Eiji Takimoto[2]　Koichi Mouri[2]
Hiroshi Matsuo[1]

1. はじめに

クラウドコンピューティングにおいて，仮想計算機間および拠点間の安全な接続を目的として，暗号化されたトンネリングの技術が利用されている．このような利用目的においては，安全性とともに高い性能が要求されるため，10Gbps 以上の高速なパケットの送受信が可能な Network Interface Controller（NIC）が利用されるようになっている．しかし，高性能な NIC を利用したとしても，CPU によるパケット処理の性能が追いつかなければ，その性能を発揮できない．特に，セキュリティのためにパケットを暗号化してトンネリングするには，CPU に大きな負担が掛かる．そのため，CPU がボトルネックとなることから，CPU によるパケット処理の性能向上が求められる．

[1] 名古屋工業大学
Nagoya Institute of Technology
[2] 立命館大学
Ritsumeikan University
[a)] h-ogawa@matlab.nitech.ac.jp

現在広く利用されているマルチコアプロセッサにおいて高速なパケット処理を実現するには，複数の CPU コアを活用することが必要である．そのために，複数の受信キューを持つマルチキュー NIC が存在する．マルチキュー NIC は，受信したパケットのフローを考慮して複数のキューに分配し，各キューに対応する CPU コアに対して割り込みを発生させることでパケット処理を複数の CPU コアに分配する．フローとは，関連するパケットの一連の流れのことである．TCP のパケットであれば，送信元と宛先の IP アドレスおよびポート番号が一致するパケットが複数送信されていれば，それらは同一セッションのパケットである可能性があるため，同一のフローとして扱う．

暗号化トンネリングを実現するためのプロトコルとして広く利用されている技術の 1 つに，Internet Protocol Security（IPsec）がある．IPsec は，IP 層のレベルで機密性や真正性を確保し，セキュリティを実現する技術である [1]．IP パケットをカプセル化することで，セキュリティのための機能を持たない IP 上のプロトコルを安全に利用

できる．

IPsec を用いてルータ間で転送されるパケットは，カプセル化前の情報が記された IP ヘッダが暗号化され，ルータ間の転送のための IP ヘッダを新たに追加した上で転送される．そのため，転送元のルータではパケットのフローを認識して並列処理されるものの，転送先のルータではすべて同一フローとみなされ，単一の CPU コアでしか処理されない．

そこで，本稿では IPsec トンネルの両端のルータが連携することにより，IPsec によるパケット転送を並列処理可能にする手法を提案する．パケットを転送するルータは，パケットを暗号化する前にパケットのフロー識別情報を生成し，暗号化後のパケットに NIC が識別可能な形で付与して転送する．転送先ルータでは，付与されたフロー識別情報を NIC が識別し，パケットを複数ある CPU コアに分配することで並列処理を実現する．

本稿では，クラウドコンピューティングにおける仮想計算機間や拠点間の暗号化されたトンネリングを並列処理の対象とする．そのため，ルータは 10Gbps 以上のパケットの送受信が可能な NIC を搭載していることを前提とする．また，ルータに接続するホストは多様なハードウェアおよびソフトウェア構成であることが予想されるため，提案手法はルータに対する変更のみで完結し，接続するホストからは透過的に利用可能なものにする．

まず，2 章で IPsec について述べる．次に，3 章でパケット処理の並列化に関する関連研究について述べる．その上で 4 章で提案手法を示し，5 章では Linux カーネルに対して提案手法を実装する．6 章では実装したシステムの性能評価について述べる．最後に 7 章でまとめる．

2. IPsec

本章では，提案手法の対象である IPsec の概要とパケット転送処理の詳細を述べる．

2.1 モードとセキュリティプロトコル

IPsec には，IP パケットをカプセル化する際のモードとしてトランスポートモードとトンネルモードの 2 つが定義されている．トランスポートモードは，元の IP パケットのペイロード部分のみをカプセル化するモードである．トランスポートモードの主な利用目的は，エンドツーエンドでの通信である．一方，トンネルモードは IP ヘッダ全体をカプセル化して IPsec ペイロードとし，新たに IP ヘッダを追加するモードである．拠点間のトンネリングを実現するための手段としては，トンネルモードを利用することが一般的であるため，本稿ではトンネルモードを対象とする．

本稿では，パケットのカプセル化を行うセキュリティプロトコルのうち，機密性を確保可能な Encapsulating Security Payload（ESP）について扱う．Authentication Header（AH）は，カプセル化時にパケットの暗号化を伴わないことから，本稿で扱う並列処理における問題が生じないためである．ただし，ESP と AH を二重に適用する場合は，ESP を単独で利用する場合と同様に問題が生じるため，本稿の対象である．

なお，本稿ではルータ間での鍵交換および接続の構築は完了しているものとし，IPsec の鍵交換のためのプロトコルである Internet Key Exchange（IKE）については扱わない．

2.2 データ構造

IPsec を構成する代表的なデータ構造に，Security Policy（SP）と Security Association（SA）がある．SP は，IPsec を利用する条件を表現するデータ構造である．SP には，IPsec を利用する IP アドレスやポート番号などの条件要素の集合であるセレクタが格納される．SP に適合するパケットが IPsec で扱う対象になる．SA は，SP に適合するパケットをどのように送信するかを表現するデータ構造である．SA には，利用するモードとセキュリティプロトコル，暗号化の鍵などのセキュリティプロトコルが必要とする情報に加えて，カプセル化後のパケットに含まれる SA を識別するための値である Security Parameter Index（SPI）が格納される．SP と SA の間には対応関係があり，一方からもう一方の情報を辿ることができる．そのため，SP と SA には両者の対応関係を表現するのに必要な情報が格納される．この情報を保持する方法は実装依存である．SP をまとめた集合を Security Policy Database（SPD），SA をまとめた集合を Security Association Database（SAD）という．

2.3 パケット転送の処理手順

本稿では，パケット転送に関わる 2 台のルータのうち，パケットをカプセル化して転送する側を転送元ルータ，パケットを受信してカプセル化を解除する側を転送先ルータと呼ぶ．

IPsec を利用したパケット転送のうち，転送元ルータで行う処理の流れを以下に示す．

(1) 送信元ホストからパケットを受信する．
(2) パケットと SPD を照合し，適合する SP を検索する．SP が存在しない場合は IPsec を利用せずに通信する．
(3) SP に対応する SA を検索し，利用するアルゴリズムと鍵を決定する．
(4) パケットをカプセル化する．
(5) 転送先ルータにカプセル化したパケットを転送する．

IPsec を利用したパケット転送のうち，転送先ルータで行う処理の流れを以下に示す．

(1) 転送元ルータからカプセル化されたパケットを受信する．

(2) カプセル化されたパケットと SAD を照合し，適合する SA を検索する．SA が存在しない場合はパケットを破棄する．
(3) SA のパラメータに基づいてカプセル化を解除する．
(4) SA に対応する SP を検索し，カプセル化を解除したパケットと SP の内容を照合する．適合しない場合はパケットを破棄する．
(5) 宛先ホストにカプセル化を解除したパケットを転送する．

3. 関連研究

本章では，パケット処理の並列化によって性能向上を図る関連研究について述べる．

3.1 マルチキュー NIC

マルチキュー NIC によって実現されるフローを考慮したパケットの並列処理機能に，Receive Side Scaling（RSS）と Flow Director がある．単一のキューしか持たない NIC では，Linux の Receive Packet Steering（RPS）を利用することで，RSS をソフトウェアで再現できる．

3.1.1 Receive Side Scaling（RSS）

マルチキュー NIC を活用した割り込みとパケット処理の分配機能を，Receive Side Scaling（RSS）という [2]．RSS におけるパケットの分配は，パケットのヘッダに基づくフローを考慮して行う．

パケットの分配先を決定する手順は，ハッシュ値の計算と Indirection Table の参照からなる．まず，パケットのヘッダのうち，フロー識別に利用する情報のタプルに対して，一方向ハッシュ関数を用いてハッシュ値を計算する．そして，ハッシュ値と分配先の対応関係が保持されている Indirection Table を参照して分配先を決定する．

処理するフローが少ない場合には，ハッシュ値が衝突しないようにタプルを構成することや，Indirection Table を変更することより，ある程度はフローを目的の CPU コアに分配できる．しかし，フローが増えた場合にハッシュ値や Indirection Table の衝突を回避できず，パケットの分配を制御できないという問題がある．

3.1.2 Flow Director

Intel の一部の高機能な NIC では，RSS に加えて Flow Director がサポートされている [3]．Flow Director では，パケット分配のためのフィルタを定義し，適合するパケットを直接指定したキューに分配できる．フィルタの定義には，いくつかある項目の 1 個以上の組み合わせが利用できる．Intel Ethernet Controller X540 において利用可能な項目を以下に示す．これらの項目について，完全一致またはマスクした範囲での一致をフィルタに設定できる．

- VLAN ヘッダ
- src および dst の IP アドレス
- src および dst のポート番号（TCP と UDP）
- プロトコルの種類（IPv4 と IPv6 および UDP と TCP と SCTP）
- パケットの先頭 64 バイトのうち任意の 2 バイト
- 対象の pool number（VT モードのみ）

Flow Director は，RSS よりもフローの判断に利用可能な項目が多く，ハッシュ関数や Indirection Table の衝突によってフローを識別できなくなる問題もない．これにより，RSS よりも柔軟にパケットの分配を制御できる．

3.1.3 Receive Packet Steering（RPS）

Linux カーネルは，Receive Packet Steering（RPS）と呼ばれる機能を持つ [4]．RPS は，単一のキューしか持たない NIC のために，RSS に似た仕組みをソフトウェアで提供する．RPS では，割り込みとパケット受信処理はハードウェアサポートなしに並列化できないため，単一の CPU コアで行う．しかし，その後のネットワークスタックの処理については，RSS と同様に処理を行う CPU コアを選択し，選択された CPU コアをプロセッサ間割り込みで起床させることで，並列処理する．RPS はパケット受信処理を並列化できないため，単一の CPU コアに負荷が集中する構造を完全には解消できないという問題があるものの，パケットの分配方法をソフトウェアで定義できるため，未知のプロトコルに対して柔軟に対応できる．

3.1.4 マルチキュー NIC による IPsec の並列処理の問題点

マルチキュー NIC を利用してパケットを並列処理するためには，NIC に認識可能な形でパケット内にフロー情報を含んでいる必要がある．IPsec トンネリングを行う場合，カプセル化されたパケットからはカプセル化前のパケットが持っていたフローを把握できず，NIC はすべてのパケットを同一フローとして扱う．このため，マルチキュー NIC においても，IPsec トンネリングの場合はすべてのパケットが単一の CPU コアで処理される．

3.2 pcrypt

pcrypt[5] は，Linux カーネルに搭載された暗号化処理を並列化する機能である．暗号化処理を複数の CPU コアに分配して並列に実行し，最後に直列化することで，入力時と同じ順序で出力する．そのため，pcrypt に対する入出力のインタフェースは pcrypt を利用しない場合と同一のままで，暗号化部分を並列化できる．

IPsec を利用したパケット転送に pcrypt を適用した場合，転送先ルータでは，図 1 に示すように，パケットの復号処理が並列化される．一方，パケットの受信やネットワークスタックは，pcrypt を適用しない場合と同様に単一の CPU コアで処理される．そのため，特定の CPU コアに負荷が集中する構造を完全に解決するものではない．加えて，暗号化の前後には処理の分配と直列化のオーバヘッ

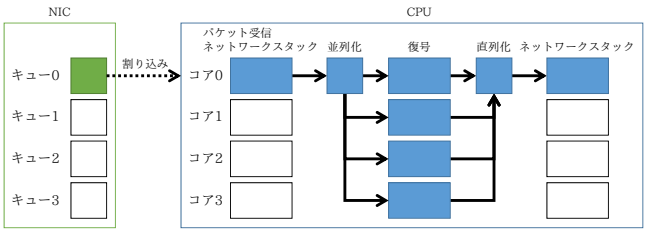

図 1　pcrypt 有効時のパケット受信処理における CPU コアの活用

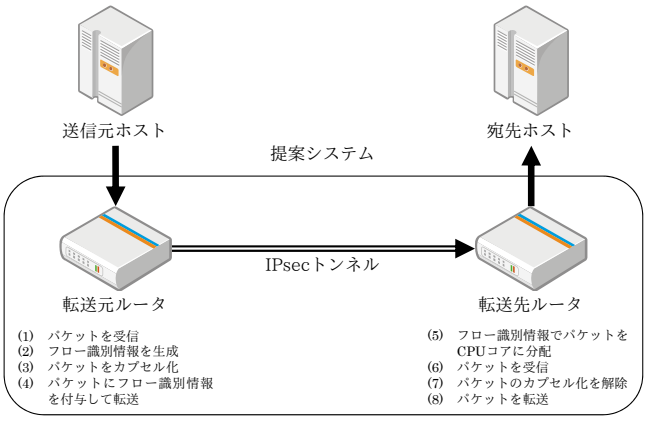

図 2　提案システムの概要

ドが必要なため，パケット処理時間の中で暗号化の占める割合が少ない小さなパケットの処理においては，性能向上の幅が小さい．

4. 提案手法

本稿では，IPsec トンネリングにおいて，パケットを並列処理できないという問題に対して，ルータ間で連携することによってパケット転送を並列化する手法を提案する．提案手法は，特定の CPU コアに負荷が偏る構造を回避し，各 CPU コアに同一の処理を行わせることが可能なため，複数の CPU コアの中で特定の CPU コアのみがボトルネックになることがない．

本章では，提案システムの概要と，転送元ルータと転送先ルータのそれぞれの動作，および提案手法が有効に働く状況について詳細を述べる．

4.1 概要

提案システムの概要を図 2 に示す．提案システムは，送信元ホストからパケットを受信して転送する転送元ルータと，転送元ルータからパケットを受信して宛先ホストにパケットを転送する転送先ルータからなる．提案システムの処理手順を以下に示す．手順 (1) から (4) までは転送元ルータでの処理であり，手順 (5) から (8) までは転送元ルータでの処理である．

(1) 送信元ホストからパケットを受信する．
(2) カプセル化前のパケットの内容に基づいてフロー識別情報を生成する．

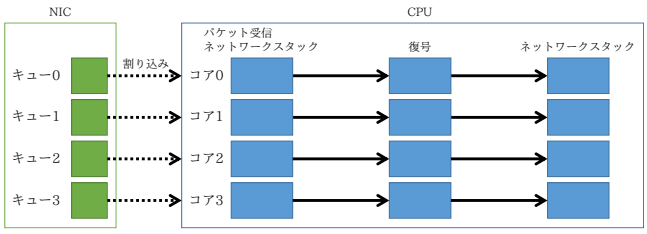

図 3　提案手法のパケット受信処理における CPU コアの活用

(3) パケットをカプセル化する．
(4) パケットにフロー識別情報を付与して転送する．
(5) 転送元ルータからのフロー識別情報でパケットを CPU コアに分配する．
(6) パケットを受信する．
(7) パケットのカプセル化を解除する．
(8) パケットを宛先ホストに転送する．

転送元ルータが送信元ホストからパケットを受信する段階では，パケットはカプセル化されていないため，NIC によってフローを識別できる．そのため，提案手法の有無に関係なく並列処理できる．一方，転送先ルータで受信するパケットはカプセル化されているため，提案手法を導入しない状態ではフローを識別できず，処理の並列化はできない，提案手法では，パケットにフロー識別情報を付与するため，転送先ルータにおいてもフローを識別して並列処理できる．提案手法は，図 3 に示すように，NIC によるパケットの分配を可能にする手法であるため，パケットの受信およびネットワークスタック，復号のすべてを並列処理できる．フロー識別情報の詳細については 4.3 節で述べる．

4.2 提案手法が有効に働く状況

提案手法は，フロー単位でパケット処理を各 CPU コアに分配することで，並列処理を実現している．そのため，フローの数が少なくとも転送先ルータの CPU コアの数以上存在しなければ，すべての CPU コアを活用することはできない．実際には，フロー識別情報の計算方法によって値が衝突することもあるため，CPU コアの数に対して十分に多いフローが必要である．しかし，本提案が想定するクラウドコンピューティングにおいては，ルータに対して多数のホストが接続することが想定されるため，実際の利用においては問題にならないと考える．

4.3 フロー識別情報

フロー識別情報は，転送先ルータでパケットを CPU コアに分配する際に利用する値である．フロー識別情報の値が異なるパケットは，確実に異なるフローのパケットであることを意味している．そのため，異なる CPU コアで処理してもパケットのリオーダリングやデータローカリティの低下に繋がらない．逆に，フロー識別情報の値が等しいパケットは，同一フローのパケットである可能性があるこ

とを意味している．実際には異なるフローであってもフロー識別情報の値が同じになる場合もあるが，その場合パケットが同一 CPU コアで逐次的に処理されるだけで，パケットの転送に不具合が生じることはない．

フロー識別情報には，2 通りの利用方法が存在する．1 つ目は，転送先ルータで利用する CPU コアの番号とする利用方法である．この場合，パケットを転送先ルータのどの CPU コアで処理するかを転送元ルータで決定した上でパケットを転送する．2 つ目は，フロー単位で識別可能な識別子とする利用方法である．ただし，フローを識別できる上限以上のフローを扱う場合は，複数のフローに対して 1 つの識別子を割り当てる．この場合，転送元ルータではフローごとに異なるフロー識別情報を割り当ててパケットを転送し，パケットをどの CPU コアで処理するかは転送先 CPU コアで決定する．

2 つの利用方法の違いは，パケット分配の偏りやパケット転送以外の負荷が発生するなどの原因により，CPU コア間での負荷の偏りが生じた場合の対処方法に影響する．フロー識別情報を CPU コアの番号とする場合，転送先ルータにパケットが到達した時点ですでにパケットを処理する CPU コアが決定されており，仮に特定の CPU コアの利用率が高い状況であったとしても，他の CPU コアにパケット処理を移譲することは難しい．これは，CPU コアに分配されたパケットには，それ以上の情報がなく，移譲する場合にフローを考慮できないためである．この手法で負荷の偏りを考慮したパケット分配を実現する場合，転送元ルータと転送先ルータが連携し，転送先ルータの負荷に応じて転送元ルータでパケットの分配比率を調整する必要がある．Ahuja らの研究 [6] では，パケットの送受信の際にホスト間で連携し，送信側で受信側の負荷を考慮する手法が提案されている．この手法は，ルータ間で連携してパケットの分配比率を調整する手法に応用可能であると考えられる．

フロー識別情報をフローの識別子とする場合，フロー識別情報と実際にパケットを処理する CPU コアの対応を決定する処理は転送先ルータに委ねられているため，自身の負荷を考慮して分配方法を制御できる．ただし，大量のフローを細かく制御するためには，高性能な NIC が必要となり，システムの性能要件が高くなる．Pesterev らの研究 [7] では，CPU コア間の負荷の偏りが生じた場合に，一時的に他の CPU コアのパケットを処理できるようにする短期的な対策と，NIC による分配を調整する長期的な対策によって負荷の偏りを解決する手法を提案している．この手法は，転送先ルータでのフローに基づく負荷の調整に応用可能であると考えられる．

5. 実装

Linux カーネルの IPsec プロトコルの実装基盤である XFRM を拡張し，提案手法を実装した．本章では，まず提案手法におけるフロー識別情報をどのように表現するかについて述べ，その後に実装した内容の詳細を述べる．負荷の偏りを考慮した高度な負荷分散については，今後の課題とする．

5.1 フロー識別情報の表現

提案手法を実現するためには，転送するパケットにフロー識別情報を付与して転送する必要がある．この時，特定の CPU コアに負荷が集中することを防ぐために，パケットの分配をハードウェアで行う方法を選択することが望ましい．パケットにフロー識別情報を付与する方法として，ESP ヘッダに含める方法と IP アドレスを利用して表現する方法が考えられる．

フロー識別情報を ESP ヘッダに含める場合，ヘッダに新たにフィールドを追加する必要がある．これにより，提案システムで利用する ESP ヘッダは既存の ESP ヘッダと互換性がなくなる問題がある．提案システムで利用する ESP に対して，本来の ESP とは異なるプロトコル番号を割り当てることで問題を回避できると考えられるが，経路上のルータで新たなプロトコルの通過許可の設定変更が必要となる問題がある．また，この方法を利用した場合，転送先ルータでは Flow Director を利用してパケットを分配する必要があるため，適用可能な環境が限定される問題がある．

IP アドレスを利用して表現する方法では，パケットの構造には手を加えず，IP アドレスのフィールドを利用してフロー識別情報を表現する．具体的には，転送元ルータと転送先ルータに複数の IP アドレスを割り当てておき，フロー識別情報の値に応じて転送に利用する IP アドレスを切り替える．転送先ルータでは，IP アドレスからフロー識別情報を取り出す．この方法を利用した場合，転送先ルータでは Flow Director と RSS のどちらでもパケットを分配できるため，幅広い環境に適用できる利点がある．これを実現するためには，通常一対一で対応する SP と SA の関係を拡張し，1 つの SP に対して複数の SA を確立できるようにした上で，使用する IP アドレスが異なる複数の SA を確立し，パケット転送元でフローに応じて SA を選択する必要がある．この方法は，通常の SP と SA が一対一で対応する場合に対してはフローによらず常に同一の SA が選択される結果となり，互換性が維持される利点がある．フロー識別情報の取りうる値の数に応じて使用する IP アドレスが増加するという問題があるが，膨大なアドレス空間を持つ IPv6 が普及することで緩和されると考える．

本稿では，既存の IPsec との互換性を維持するのが容易であることと，幅広い環境に適用可能なことから，フロー識別情報の表現方法として IP アドレスを利用する方法を選択した．ただし，クラウドコンピューティング向けの VPN ルータとネットワークスイッチでは，高機能な NIC を利用することが想定されるため，本稿では Flow Director

を用いたパケット分配のみを実装した．

5.2 XFRM の拡張

XFRM は，Linux カーネルに搭載されたパケット変換フレームワークである [8]．XFRM は，主に xfrm_policy と xfrm_state の 2 つのデータ構造からなり，それぞれ IPsec における SP と SA に相当する．xfrm_policy は，対応する xfrm_state を特定するための情報を xfrm_tmpl として含んでいる．

提案システムでは，1 つの SP に複数の SA を対応させられるようにするために，xfrm_policy が複数の xfrm_tmpl を保持できるよう拡張した．それに伴い，転送元ルータでは，登録された xfrm_tmpl を利用して xfrm_state を取り出す際に，複数登録されていた場合はどれか 1 つを選択するように変更した．転送先ルータでは，xfrm_state から対応する xfrm_policy を検索する際に，複数ある xfrm_tmpl のうちどれかにマッチすれば対応する xfrm_policy であるとみなすように条件を変更した．

5.3 xfrm_state の選択とキャッシュ

xfrm_state の選択結果は，カーネル内に一定期間キャッシュされる．キャッシュの単位は IP アドレス，プロトコル番号，ポート番号などである．キャッシュが有効である期間は，フローに対して同一の xfrm_state を選択することが保証されるため，xfrm_state を選択するアルゴリズムはフローを考慮する必要はなく，複数ある xfrm_state を等しい確率で選択して分配できれば問題ない．本稿では，カプセル化を行う CPU コアの番号に応じて xfrm_state を選択する方法を採用した．

5.4 パケットの分配

Flow Director を利用してパケットを分配する場合，送信元または転送先の IP アドレスを条件にしたフィルタを，フロー識別情報が取りうる値の数だけ事前に定義する．RSS を利用してパケットを分配する場合，ハッシュ値や Indirection Table を参照した結果が衝突することを考慮する必要がある．衝突が発生した場合，そのフローは区別できなくなり，同じ CPU コアにしか分配できない．衝突を回避するためには，ハッシュ値が衝突しない IP アドレスの組み合わせを用意した上で，Indirection Table の該当する位置に異なる分配先を記述する．RSS を利用する場合，ルータに割り当て可能な IP アドレスに制約が生じるため，Flow Director をサポートする NIC を利用するのであれば，RSS を選択する必要はない．

6. 評価

提案手法の有効性を確認するため，実装した提案システムを評価した．評価には，片方向の転送性能を測定するマイクロベンチマークと，双方向の転送性能を測定するアプリケーションによるベンチマークを用いた．

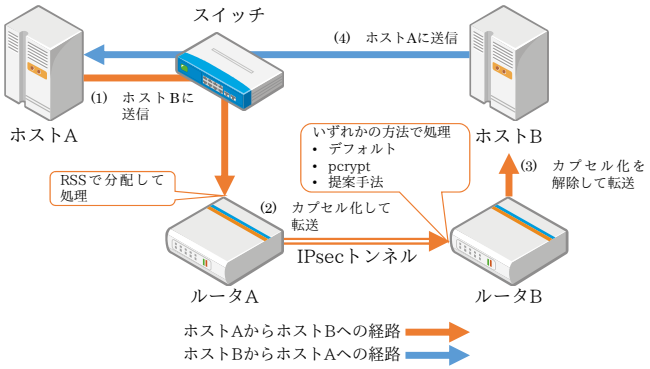

図 4 マイクロベンチマークのネットワーク構成

表 1 ホスト A およびルータ A，ルータ B のスペック

OS	Arch Linux (Linux Kernel 3.14.1)
CPU	Intel Core i7 4770（4 コア，3.4GHz）
メモリ	8GB
NIC	Intel Ethernet Converged Network Adapter X540-T2

表 2 ホスト B のスペック

OS	Arch Linux (Linux Kernel 3.14.1)
CPU	Intel Core i7 3930K（6 コア，3.2GHz）
メモリ	16GB
NIC	Intel Ethernet Converged Network Adapter X540-T2

6.1 マイクロベンチマーク

マイクロベンチマークでは，本稿で想定する多数のフローが存在する場合を再現して評価した．マイクロベンチマークによる評価を行ったネットワークの構成を図 4 に示す．ホスト A およびルータ A，ルータ B のスペックを表 1 に，ホスト B のスペックを表 2 に示す．スイッチには NETGEAR XS712T を利用した．

片方向の転送性能を評価するため，図 4 におけるホスト A からホスト B に送信するパケットだけをトンネリングし，ホスト B からホスト A へのパケットはトンネリングせずに送信するように設定した．ホスト A とホスト B 間でのパケットの往復は以下の手順で行う．

(1) ホスト A は宛先をホスト B とするパケットをルータ A に向けて送信する．
(2) ルータ A はパケットをカプセル化してルータ B に転送する．
(3) ルータ B はパケットのカプセル化を解除してホスト B に転送する．
(4) ルータ B はホスト A にパケットを送信する．

ルータ A がホスト A からパケットを受信する場合，パケットはカプセル化されていないため，ルータ A では RSS を利用してパケットを並列処理できる．一方，ルータ B がルータ A からパケットを受信する場合，パケットはカプセル化されているため，RSS では並列処理できない．そこ

で，この部分について，以下の 3 つの手法でそれぞれベンチマークを行い，性能を比較した．

- 並列処理を行わないデフォルトの場合（デフォルト）
- パケット受信部分はそのままで，パケットの復号のみを pcrypt で並列処理する場合（pcrypt）
- パケットを分配して並列処理する提案手法を導入した場合（提案手法）

ホスト A とホスト B でベンチマークプログラムのクライアントとサーバをそれぞれ起動し，ベンチマーク実行中のルータ B のパケット送信のスループットを測定した．サーバは，ワーカスレッドを 6 本起動し，クライアントからの接続を待ち受ける．ソケット作成時に SO_REUSEPORT オプションを指定することで，すべてのワーカスレッドが単一のポートに対して待ち受けられる．ワーカスレッドは epoll システムコールを利用したイベント駆動モデルで実装しており，各ワーカスレッドが複数のコネクションを非同期に並列処理できる．クライアントは，ワーカスレッドを 4 本起動し，各ワーカスレッドがそれぞれサーバに対して 64 本の TCP コネクションを確立する．各コネクションで指定したサイズのメッセージ送信を 120 秒間繰り返す．ただし，10 回メッセージを送信するごとに TCP コネクションを切断し，再度接続し直す処理を行う．ワーカスレッドはサーバと同様に epoll システムコールを利用したイベント駆動モデルで実装した．

暗号化には AES（128bit）の AES-NI を利用する実装，認証には HMAC-SHA1 の SSSE3 を利用する実装を選択した．すべてのホストとルータにおいて，CPU のクロック周波数は最大に固定し，Hyper-Threading はオフにした．使用するすべての NIC の MTU は 9000 に設定し，パケット処理のオフロード機能はオフにした．ただし，実験環境では IPsec トンネリング区間においてパケットが 1500 バイト単位で分割されることを確認している．

送信するメッセージサイズを 64 バイトから 16KB まで変化させた時のマイクロベンチマークの実行結果を図 5 に示す．図 5 より，pcrypt は，メッセージサイズが 512 バイト以上の場合にデフォルトよりも高いスループットを示したものの，256 バイト以下の場合にはデフォルトを下回った．これは，メッセージサイズが小さい場合には全体の処理量に占める暗号化の割合が小さいために，並列処理によって得られるメリットが小さいことと，並列処理に必要な並列化と直列化に一定のオーバヘッドが存在することが原因であると考えられる．一方，提案手法は，測定したすべてのメッセージサイズにおいて pcrypt およびデフォルトと比べて高いスループットを示した．提案手法では，パケット受信処理と復号およびネットワークスタックのすべての処理が並列処理されるため，トラフィックの傾向が変化しても並列処理の効果を得やすい構造であることと，ハードウェアの段階で処理が CPU コアに分配されるため，

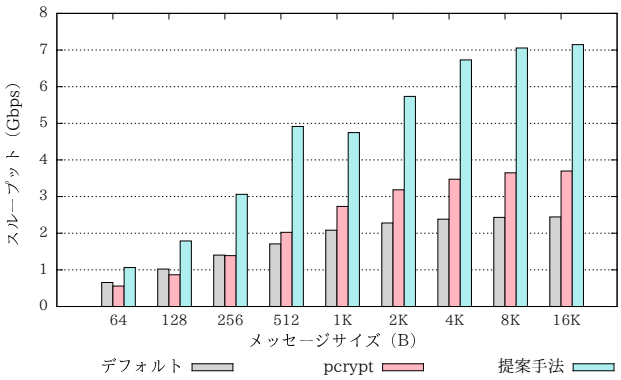

図 5　マイクロベンチマークの実行結果

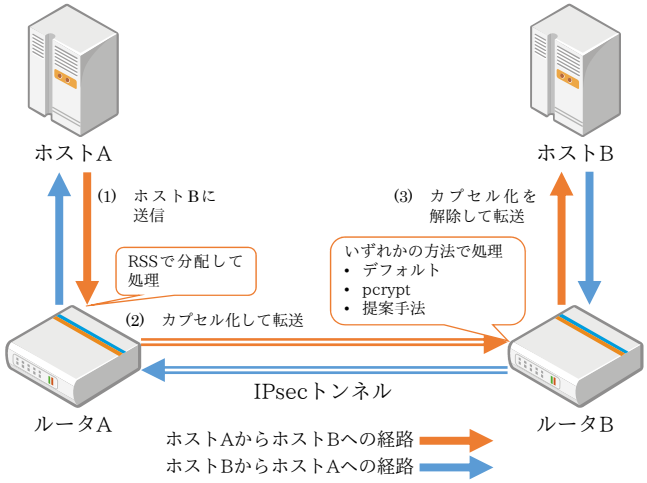

図 6　アプリケーションによるベンチマークのネットワーク構成

並列処理に伴うオーバヘッドがほとんど存在しないことが理由であると考えられる．

6.2　アプリケーションによるベンチマーク

Web サーバである nginx を対象としてベンチマークを行った．評価を行ったネットワークの構成を図 6 に示す．各機器のスペックは表 1 および表 2 と同様である．

マイクロベンチマークの場合とは異なり，図 6 のネットワークでは，より実際の利用環境に近い状況で評価するため，ホスト A からホスト B に送信するパケットとホスト B からホスト A に送信するパケットの両方をトンネリングの対象とした．図 6 中のホスト A からホスト B へのパケットの送信は以下の手順で行う．ホスト B からホスト A へ送信する手順も同様である．

(1) ホスト A は宛先をホスト B とするパケットをルータ A に向けて送信する．
(2) ルータ A はパケットをカプセル化してルータ B に転送する．
(3) ルータ B はパケットのカプセル化を解除してホスト B に転送する．

ルータ A がホスト A からパケットを受信する場合およびルータ B がホスト B からパケットを受信する場合，パ

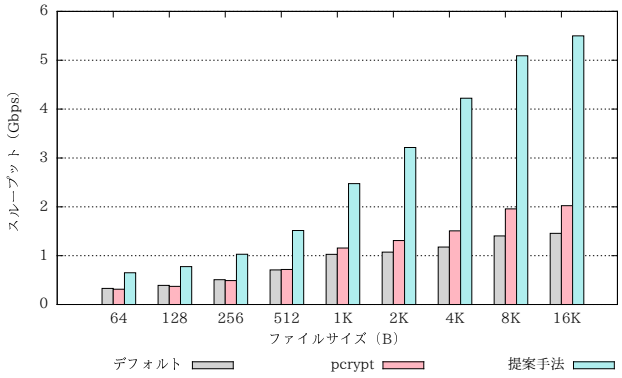

図 7 アプリケーションによるベンチマークの実行結果

ケットはカプセル化されていないため，RSS を利用してパケットを並列処理できる．一方，ルータ B がルータ A からパケットを受信する場合およびルータ A がルータ B からパケットを受信する場合，パケットはカプセル化されているため，RSS では並列処理できない．そこで，この部分について，マイクロベンチマークと同様に 3 つの手法でそれぞれベンチマークを行い，性能を比較した．マイクロベンチマークとの違いは，ルータ A と B の両方で暗号化と復号が行われる点である．

ホスト B で nginx[9] を起動し，ホスト A で HTTP のベンチマークプログラムである ab[10] を実行し，ab の出力するスループットを計測した．nginx はワーカスレッドを 6 本起動し，静的なファイルを配信する．ファイル配信時にディスク I/O が発生することを防ぐため，一度配信したファイルはメモリ上にキャッシュするように設定した．さらに，計測前にファイルにアクセスすることで，計測中は常にキャッシュが働くようにした．Keep-Alive を有効にし，10 回のリクエストまではコネクションを切断しないようにした．ab は，256 並列でホスト B の nginx に対して指定した同一ファイルのリクエストを 10,000,000 回繰り返す．その他の設定は 6.1 節と同様である．

リクエストするファイルサイズを 64 バイトから 16KB まで変化させた時のベンチマークの実行結果を図 7 に示す．図 7 より，アプリケーションによるベンチマークにおいても，マイクロベンチマークと同様の傾向が確認できた．このことから，提案手法は多数のフローが双方向で転送される実際の環境においても，高いスループットを達成することが期待できる．

7. まとめ

本稿では，IPsec トンネリングを並列処理するために，IPsec トンネルの両端のルータが連携する手法を提案した．提案手法は，パケット転送元のルータがカプセル化前のパケットのフローからフロー識別情報を生成し，カプセル化後のパケットに付与して転送することで，転送先のルータでフローに基づいたパケットの並列処理を可能にする．ルータに複数の IP アドレスを割り当てて IPsec の SA を多重化し，フローに応じて使用する IP アドレスを切り替えることでフロー識別情報を付与する手法の実装を行った．実装したシステムについて，マイクロベンチマークおよびアプリケーションによるベンチマークによって性能を評価し，提案手法は既存手法である pcrypt と比較して高速なパケット転送が可能であることを確認した．今後は，CPU 間で負荷の偏りが生じた場合にも対応可能な高度な負荷分散機能の導入を検討する予定である．

参考文献

[1] 谷口　功，水沢紀子：マスタリング TCP/IP IPsec 編，オーム社 (2005).
[2] Introduction to Receive Side Scaling: `http://msdn.microsoft.com/en-us/library/windows/hardware/ff556942(v=vs.85).aspx`.
[3] Intel Ethernet Controller X540 Datasheet: `http://www.intel.co.jp/content/dam/www/public/us/en/documents/datasheets/ethernet-x540-datasheet.pdf`.
[4] Herbert, T. and de Bruijn, W.: Scaling in the Linux Networking Stack, `https://www.kernel.org/doc/Documentation/networking/scaling.txt`.
[5] Klassert, S.: Parallelizing IPsec: switching SMP to 'On' is not even half the way, `http://www.strongswan.org/docs/Steffen_Klassert_Parallelizing_IPsec.pdf`.
[6] Vishal Ahuja, M. F. and Ghosal, D.: Cache-Aware Affinitization on Commodity Multicores for High-Speed Network Flows, *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, pp. 39–48 (2012).
[7] Aleksey Pesterev, Jacob Strauss, N. Z. and Morris, R. T.: Improving Network Connection Locality on Multicore Systems, *Proceedings of the 7th ACM european conference on Computer Systems*, pp. 337–350 (2012).
[8] Rosen, R.: *Linux Kernel Networking: Implementation and Theory*, Apress (2013).
[9] nginx news: `http://nginx.org/`.
[10] ab - Apache HTTP server benchmarking tool: `http://httpd.apache.org/docs/2.2/programs/ab.html`.