マルチキューNICを用いたIPsecの並列化手法の実装と評価

小川　拡[†a)]　齋藤　彰一[†]　安井　裕亮[†]　川島　龍太[†]
瀧本　栄二[††]　毛利　公一[††]　松尾　啓志[†]

Implementation and Evaluation of IPsec Parallelizing Method with Multi-Queue NIC

Hiromu OGAWA[†a)], Shoichi SAITO[†], Yusuke YASUI[†], Ryota KAWASHIMA[†], Eiji TAKIMOTO[††], Koichi MOURI[††], and Hiroshi MATSUO[†]

あらまし　本論文では，IPsecを高速化するための並列処理手法を提案する．通常，IPsecで広く利用されているEncapsulating Security Payload（ESP）とトンネルモードの組み合わせでは，転送するパケットをカプセル化する際にヘッダを含めて暗号化するため，転送先のルータでは受信したパケットからパケットフローを認識できず，マルチキューNICのもつ並列処理機能を活用できない．提案手法では，トンネルの両端のルータが連携することでこの問題を解決する．パケットを転送する際に，カプセル化前のパケットからフロー識別情報を生成してカプセル化後のパケットに付与することで，転送先のルータにおけるパケットフローの認識を可能にし，マルチキューNICを活用した並列処理を実現する．提案システムをLinux kernel 3.14.1に実装し，4コアのルータを利用してパケット転送性能をデフォルトのカーネルと比較したところ，64 Byteから128 KByteまでのメッセージを送信する単純なTCPパケット転送性能の比較では100%から204%性能が向上した．

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

1. まえがき

データセンタ内のカプセル化を伴う通信及びデータセンタ間のVirtual Private Network（VPN）による通信において，仮想計算機間及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，10 Gbps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付かなければ，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．

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もつマルチキュー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アドレス及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，それらは同一コネク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一つに，Internet Protocol Security（IPsec）[1]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

† 名古屋工業大学，名古屋市
Nagoya Institute of Technology, Gokiso-cho, Showa-ku, Nagoya-shi, 466–8555 Japan
†† 立命館大学，草津市
Ritsumeikan University, 1–1–1 Noji-higashi, Kusatsu-shi, 525–8577 Japan
a) E-mail: cloud@mail.ssn.nitech.ac.jp

を実現する技術である．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もたない 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，マルチキュー NIC を用いた場合，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全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，本論文では，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マルチキュー NIC を用いて並列処理可能にする手法を提案する [2](注1)．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．

本論文では，データセンタ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 10 Gbps 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及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．

本論文の構成は次のとおりである．まず，**2.** で IPsec について述べる．次に，**3.** 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.** で提案手法を示し，**5.** でフロー識別情報の実現方法を検討したのち，**6.** では提案手法の Linux カーネルへの実装について述べる．**7.** 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 **8.** 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二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，本論文ではトンネルモードを対象とする．

本論文では，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，本論文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，本論文の対象である．

なお，本論文ではルータ間での鍵交換及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．

(注1)：本論文は，文献 [2] の内容に加えて，性能評価を加筆修正したものである．

2.3 パケット転送の処理手順

本論文では，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，IPsec におけるパケット処理の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．

3.1.1 Receive Side Scaling（RSS）

Receive Side Scaling（RSS）[3] は，マルチキュー NIC によって実現される割り込みとパケット処理の分配機能の一つである．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われる．フローを考慮しない場合，関連するパケットが異なるコアに分配され，リオーダリングによって性能が低下する可能性がある．

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．

3.1.2 Flow Director

Intel の一部の高機能な NIC では，RSS に加えて Flow Director [4] がサポートされている．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，幾つかある項目の 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．

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

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．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 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．転送元ルータが送信元ホストからパケットを受信した際，パケットはまだ暗号化されていないため，マルチキュー NIC の機能を利用してパケットを並列処理できる．一方，転送先ルータが転送元ルータから受信するパケットはカプセル化されているため，カプセル化前のパケットがもっていたフローを把握できず，NIC は全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 の場合は全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

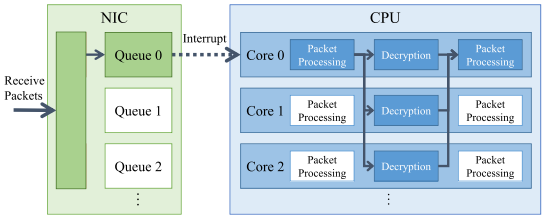

図 1　pcrypt 有効時のパケット受信処理における CPU コアの活用
Fig. 1　CPU utilization for receiving packets in IPsec with pcrypt.

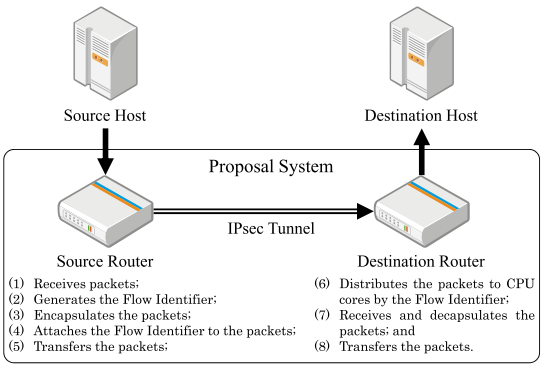

図 2　提案システムの概要
Fig. 2　An architecture of the proposal system.

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，パケットの復号処理のみが並列化される．一方，パケットの受信やネットワークスタックは，pcrypt による並列化の対象ではない．これらの処理は，転送元ルータにおいてはマルチキュー NIC によって並列処理できるものの，転送先ルータでは図 1 に示すように並列処理できない．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，暗号化の前後で処理の分配と直列化のオーバヘッドが生じるため，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．

3.3　ハードウェア実装

IPsec を高速化する研究に，Liu らによるプログラマブルマルチコアプロセッサを利用する研究 [6]，Niu らによるネットワークセキュリティプロセッサを利用する研究 [7]，及び Driessen らによる FPGA に実装する研究 [8] がある．これらは新たに専用のハードウェアを追加することによる高速化であり，本研究とはアプローチが異なる．しかし，本研究とこれらの研究は補完関係にあり，同時に使用することによってより高速な IPsec 通信を実現できると考える．

4.　提 案 手 法

本論文では，IPsec 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，各 CPU コアに独立して処理を行わせることで，特定の CPU コアに負荷が偏る構造を回避しており，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，提案手法の概要と，転送元ルータと転送先ルータのそれぞれの動作，及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

提案手法の概要を図 2 に示す．提案手法は，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．提案手法の処理手順を以下に示す．手順（1）から（5）までは転送元ルータでの処理であり，手順（6）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．
（3）　パケットをカプセル化する．
（4）　カプセル化されたパケットにフロー識別情報を付与する．
（5）　パケットを転送先ルータに転送する．
（6）　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．
（7）　パケットを受信し，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

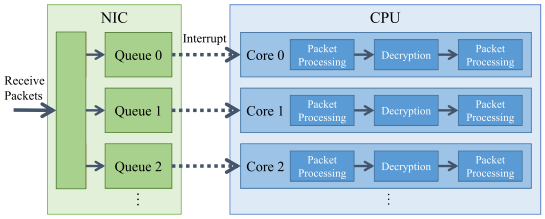

図 3　提案手法のパケット受信処理における CPU コアの活用
Fig. 3　CPU utilization for receiving packets in the proposal method.

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ことで，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，パケットの受信及びネットワークスタック，復号の全てを並列処理できる．フロー識別及びフロー識別情報の詳細については **4.3** 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 CPU コアの数を下回る場合には，全ての 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，本提案が想定するデータセンタにおいては，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　フローの識別

一般的に，パケットフローを判断する際には主にヘッダの 5 tuple（トランスポート層プロトコル，送信元 IP アドレス，宛先 IP アドレス，送信元ポート番号，宛先ポート番号）が利用される．しかし，これらの情報を平文の状態でパケットに付与することは，機密性の低下につながる．そのため，転送元のルータでカプセル化前のパケットからパケットフローを判断し，その結果を示す値のみをフロー識別情報としてパケットに付与する．

転送先ルータでは，フロー識別情報の値に基づいてパケットを CPU コアに分配する．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．

5.　フロー識別情報

本章では，提案手法を実装するにあたり，フロー識別情報の表現及び利用についての検討内容について述べる．

5.1　フロー識別情報の表現方法

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．フロー識別情報は，転送先ルータにおけるパケット分配処理が CPU コアで行われることがないように，NIC で分配が可能な形式で付与することが望ましい．転送先ルータの NIC による分配が可能なフロー識別情報の付与方法として，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．この方法には，提案システムで利用する ESP ヘッダが既存の 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この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になるという別の問題が発生する．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 アドレスそのもの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 アドレスをフロー識別情報とする．この方法を利用した場合，転送先ルータでは RSS と Flow Director 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．この方法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，一

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，転送元ルータでフローに応じて 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はフローによらず常に同一の SA が選択されるため，互換性が維持される利点がある．一方，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もつ IPv6 が普及することで緩和されると考える．

本提案では，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．

5.2 フロー識別情報によるパケット分配

5.1 で述べたように，IP アドレスで表現されたフロー識別情報は，転送先ルータにおける各 CPU コアへのパケット分配に，RSS と Flow Director の両方が対応可能である．RSS は，Flow Director と比較して幅広い種類の NIC に実装されている機能であるため，RSS を利用してパケットを分配する方法は動作環境が多い利点がある．しかし，RSS ではハッシュ値や 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，それら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．

一方，Flow Director を利用してパケットを分配する方法では，パケットはフィルタの条件と完全一致で比較される．これにより，利用する IP アドレスに制約が生じない利点がある．しかし，RSS と比較して対応する NIC が少なく，10 Gbps 以上での送受信が可能な高性能なものに限定される問題がある．

データセンタにおける 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，本提案では Flow Director を利用してパケットを分配する方法を選択した．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．

5.3 フロー識別情報の割り当て単位

フロー識別情報の割り当て単位として，転送先ルータで利用する CPU コア単位とする方法と，各フロー単位とする方法の二つが考えられる．一つ目の転送先ルータで利用する CPU コア単位とする方法では，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．二つ目の各フロー単位とする方法では，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する．パケットをどの CPU コアで処理するかはフローごとに転送先ルータが決定する．なお，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一つの識別子を割り当てる．二つの割り当て方法の違いは，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って 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 コア単位とする場合，転送先ルータにパケットが到達した時点で既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，処理する CPU コア以外の情報がなく，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 [9] 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，CPU コア単位での割り当て方法のルータ間での分配比率調整に応用可能であると考える．

フロー識別情報をフロー単位とする場合，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，転送先ルータ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 [10] 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，フロー単位での割り当て方法の負荷調整に応用可能であると考える．

本提案では，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．したがって，どちらの方法を選択しても性能上の大きな違いはないと考える．そのため，フロー識別情報の情報量を少なくできる CPU コア単位を選択した．

5.4 フロー識別情報の割り当て方法

CPU コア単位でのフロー識別情報の割り当てにより，転送先ルータの各コアの負荷はコアに割り当てられるトラヒックに依存する．そのため，全コアを偏りなく使用するにはコアに割り当てるフローの数を均一にする必要がある．加えて，パケットのリオーダリングを避けるために，同一のフローに対しては同一の CPU コアを使う必要がある．これらの条件を満たす転送元ルータにおけるフロー識別情報の割り当て方法として，RSS での分配結果を利用する方法と，ソフトウェアでパケットのフローを判断する方法が考えられる．

RSS での分配結果を利用する方法では，転送元ルータが送信元ホストからパケットを受信した際の RSS での分配結果に応じてフロー識別情報を割り当てる．RSS では，ハッシュ関数と Indirection Table により，コア間のフロー数がおおむね平等になるように分配されるため，その結果をフロー識別情報とすることで，各コアの負荷がおおむね均等になることが期待できる．しかし，分配数が転送元ルータのコア数となるため，転送元ルータと転送先ルータの CPU のコア数が異なる場合には適用できない．

ソフトウェアでパケットのフローを判断する方法では，転送元ルータ上でソフトウェアによってパケットのヘッダからフローを判断し，フロー識別情報を割り当てる．フローを判断する処理は，RSS がハードウェアで行っているものと同様のものをソフトウェア的に行う．各フローにフロー識別情報を割り当てる処理は，転送先ルータの CPU のコア数を考慮して行う．これにより，転送元ルータと転送先ルータの CPU のコア数が異なっている場合であっても，転送先ルータの各コアに対応したフロー識別情報を適切に割り当てられる．しかし，パケットのフローを判断する処理が増えるため，フロー識別情報の割り当てオーバヘッドは RSS での分配結果を利用する方法と比べて大きい．

本提案では，転送元ルータと転送先ルータの CPU コア数が異なる場合にも対応可能である点から，ソフトウェアでパケットのフローを判断する方法を採用した．なお，二つの方法について **7.1** と同様の内容でパケット転送性能を比較する予備評価を実施したところ，有意な性能差は見られなかった．そのため，ソフトウェアでパケットのフローを判断することによるオーバヘッドはパケット転送処理全体で見れば十分に小さいといえる．

6. 実　　装

Linux カーネルの IPsec プロトコルの実装基盤である XFRM [11] を拡張し，提案手法を実装した．実装は，複数の IP アドレスを切り替えることでフロー識別情報を付与したパケット転送を実現する機能の XFRM への追加と，転送先ルータにおいてフロー識別情報に基づいてパケットを分配するフィルタの NIC への追加からなる．なお，転送元ルータでは，送信元ホストからのパケットを CPU コアに分配するために RSS を設定した．これにより，転送元ルータでの転送処理は，各 CPU コアで並列に実行される．

6.1 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．XFRM は，図 4 に示すように，主に `xfrm_policy` と `xfrm_state` の二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．

提案手法では，複数の IP アドレスを切り替えるために，一つの SP に対して複数の SA を保持できるようにする．提案システムでは，これを XFRM 上で実現するために，`xfrm_policy` が複数の `xfrm_tmpl` を保持できるよう図 5 に示すように拡張した．それに伴い，転送元ルータでは，登録された `xfrm_tmpl` を

図 4　XFRM の主要なデータ構造
Fig. 4　Data structures of XFRM.

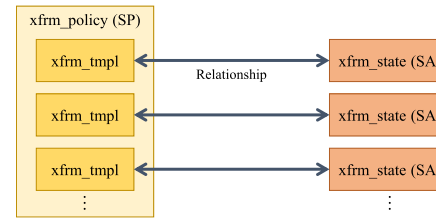

図 5　拡張した XFRM の主要なデータ構造
Fig. 5　Data structures of extended XFRM.

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 5.4 で述べたソフトウェアによる RSS と同様の処理でパケットのフローを判断し，一つの xfrm_tmpl 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．

6.2 フロー識別情報によるパケットの分配

転送先ルータでは，フロー識別情報である IP アドレスに応じてパケットを各コアに対応するキューに分配する．分配する機能は，転送先ルータの NIC に対して Flow Director のフィルタを追加することで実現した．フィルタは，送信元ルータまたは転送先ルータの IP アドレスを条件としたものをフロー識別情報の取りうる値の数だけ設定した．各フィルタの分配先は，それぞれ異なるキューにすることで，フロー識別情報の値に応じてキューが切り替わるようにした．

7. 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，単純なトラヒックによって転送性能を評価するベンチマークと，性質の異なる複数のトラヒックを混在させたより実際の環境に近いベンチマークを用いた．単純なトラヒックによるベンチマークには，片方向の転送性能を計測するマイクロベンチマークと，双方向の転送性能を計測するアプリケーションによるベンチマークを用いた．

7.1 マイクロベンチマーク

マイクロベンチマークでは，本論文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．

7.1.1 評 価 方 法

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 6 に示す．スイッチには NETGEAR XS712T を利用した．ルータ及びホストのスペックを表 1 に示す．なお，ルータ及びホストのカーネルは，CPU 時間を高精度に計測するために CONFIG_VIRT_CPU_ACCOUNTING を有効にした．

片方向の転送性能を評価するため，図 6 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，TCP の ACK などのホスト 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．

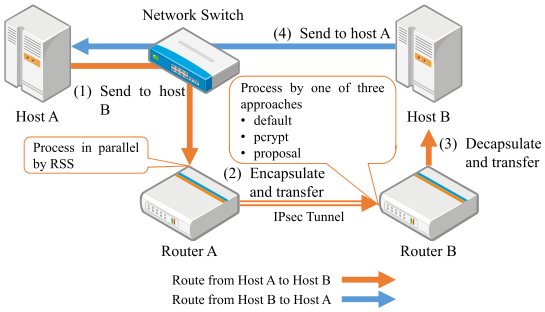

図 6　マイクロベンチマークのネットワーク構成
Fig. 6　Network configuration of the micro-benchmark.

表 1　ルータとホストのスペック
Table 1　Specifications of routers and hosts.

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

（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三つの手法でそれぞれベンチマークを実行し，比較した．

- 並列処理を行わないデフォルトの場合（default）
- 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（proposal）

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，処理中のスループットと CPU 使用率を計測した．サーバは，ワーカスレッドを 4 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，全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 ずつ，計 256 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．メッセージのサイズは 64 Byte から 128 KByte までを指定する．

なお，スループットと CPU 使用率の計測は，三つの手法を切り替えているルータ B 上にて行った．スループットは，ルータ B からホスト B に向けて転送されたパケットのデータ量を合計し，計測時間で割って算出した．CPU 使用率は，計測中の全ての CPU 時間のうち，タスク実行待ち状態になっていた時間を除いた残りの時間が占める割合とした．

暗号化には AES（128 bit）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．全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 は無効にした．パケット処理のオフロード機能は無効とし，使用する全ての NIC の MTU は 1500 Byte に設定した．

7.1.2 性能の比較

マイクロベンチマークを実行したときのスループットと CPU 使用率を図 7 に示す．CPU 使用率は 4 コア合計値を最大 100%としているため，各コアの値は最大 25%である．図 7 より，pcrypt は，メッセージサイズが 512 Byte 以上の場合に default よりも高いスループットを示したものの，256 Byte 以下の場合には default 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，並列処理に必要な並列化と直列化によるオーバヘッドが顕著化したことが原因であると考えられる．一方，proposal は，計測した全てのメッセージサイズにおいて pcrypt 及び default と比べて高いスループットを示しており，提案手法はデフォルトのカーネル及び既存手法と比較して高速なパケット転送を実現しているといえる．また，default の CPU 使用率は 1 コア分に相当する 25%に留まっているのに対し，pcrypt と proposal は 25%を超えていることから，default はシングルコア，pcrypt と proposal はマルチコアで動作していることが期待できる．なお，CPU 使用率の詳細については次項で述べる．

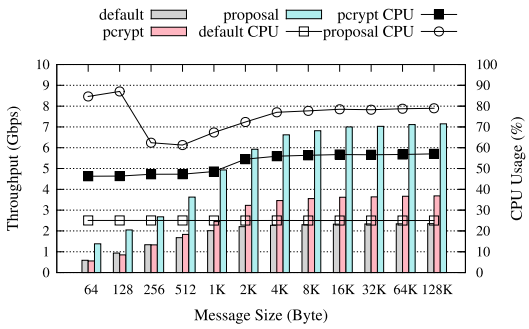

図 7 マイクロベンチマークにおけるスループットと CPU 使用率

Fig. 7 Throughput and CPU usage of the micro-benchmark.

7.1.3 マルチコア CPU の活用状況の確認

マイクロベンチマーク実行時の CPU 使用率の内訳を図 8 に示す．各メッセージサイズごとに，四つの CPU コアそれぞれの内訳を示し，0 から 3 はコア番号を示している．CPU 使用率の usr はユーザ空間，sys はシステム，wai は I/O 完了待ち，hiq は割り込み，siq はソフト割り込みで消費された CPU 時間を意味する．なお，usr，wai 及び siq による CPU 時間の消費は，ベンチマーク実行中にはほぼ 0%だった．

図 8 (a) より，default では core 0 のみが動作しているのに対して，図 8 (b) の pcrypt と図 8 (c) の proposal では複数の CPU コアが動作していることを確認した．default では，ほぼ全ての CPU 時間が hiq の処理によって消費されている．pcrypt では，core 0 と core 2 での hiq の処理以外に，全ての CPU コアで sys の処理によっても CPU 時間が消費されている．これは，pcrypt では default や proposal とは異なり，パケットの復号処理を各 CPU コアに作成したワーカスレッドに移譲しているためである．proposal では，default と同様にほぼ全ての CPU 時間が hiq の処理によって消費されている．提案手法は，NIC によってパケットを各 CPU コアに分配し，その後の CPU によるパケット処理は通常時と同様に各 CPU コアが独立して行う．そのため，default と同様の処理時間の内訳でマルチコアで動作している．

7.1.4 proposal における CPU 使用率の確認

proposal のマイクロベンチマーク実行時に送信され

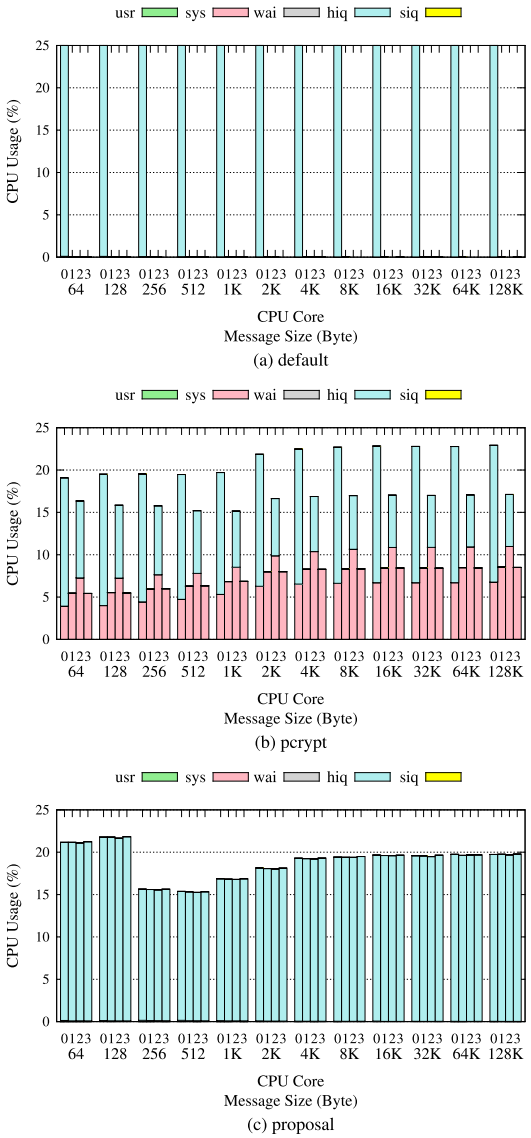

図 8　マイクロベンチマークにおける CPU 使用率の内訳
Fig. 8　Breakdown of CPU usage of the micro-benchmark.

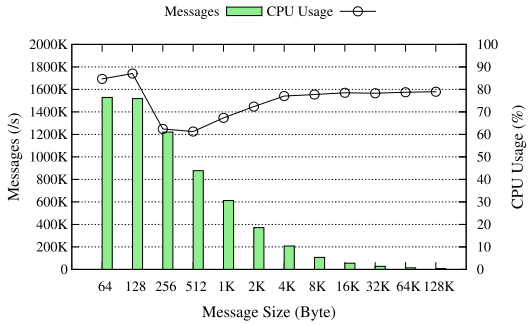

図 9　マイクロベンチマークにおける proposal のメッセージ数と CPU 使用率
Fig. 9　The number of messages and CPU usage of proposal of the micro-benchmark.

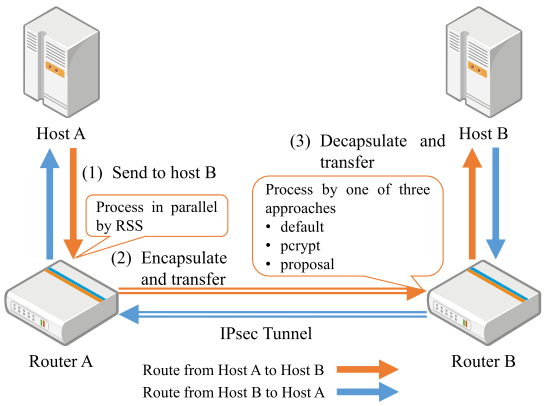

図 10　アプリケーションによるベンチマークのネットワーク構成
Fig. 10　Network configuration of the benchmark using applications.

たメッセージ数と CPU 使用率を図 9 に示す．図 7 及び図 9 より，proposal ではメッセージサイズが 256 Byte から 512 Byte の場合に CPU 使用率が低下した．512 Byte 以下の小さいメッセージサイズにおいては，ネットワークスタックの処理負荷が相対的に高まることから，メッセージ数と CPU 使用率に関係が見られる．一方，512 Byte を超える大きいメッセージサイズにおいては，復号の処理負荷が相対的に高まることから，スループットと CPU 使用率に関係が見られる．メッセージサイズによってパケット処理の負荷の内訳が変化することから，CPU 使用率に影響を与える要素が変化していると考えられる．

7.2　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 [12] 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 10 に示す．各機器のスペックは図 1 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 10 のネットワークでは，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 10 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．

（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．

ホスト B からホスト A へ送信する手順も同様である．

ルータ A がホスト A からパケットを受信する場合及びルータ B がホスト 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及びルータ 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三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 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 [13] を実行し，ab の出力するスループットをグッドプットとして計測した．nginx はワーカスレッドを 4 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．更に，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 120 秒間繰り返す．その他の設定は **7.1** と同様である．

リクエストするファイルサイズを 64 Byte から 128 KByte まで変化させたときのベンチマークのグッドプットを図 11 に示す．図 11 より，アプリケーションによるベンチマークにおいても，全てのファイルサイズにおいて，proposal は default 及び pcrypt と比較して高速にパケットを転送できている．

7.3 複数のトラヒックが混在するベンチマーク

複数のトラヒックが混在するベンチマークでは，**7.1** のマイクロベンチマークをベースに，送信するメッセージのサイズと 1 回の TCP コネクションあたりのメッセージ送信回数の異なる 2 種類のトラヒックを混在させて評価した．評価に利用したトラヒックのパターンを表 2 に示す．Small は，制御のために細かなメッセージの送信を繰り返す状況を想定したトラヒックである．Large は，ファイルなどのまとまった量のデータを送信する状況を想定したトラヒックである．TCP のコネクションを確立する際に，コネクション単位で Small か Large のどちらのトラヒックとして振る舞うか選択し，トラヒックのパターンに基づいて通信する．トラヒックの選択は，あらかじめ設定した Small と Large のトラヒックの比率に従って行う．トラヒックの比率は，Small と Large のトラヒックで送信されるパケットのデータ量がほぼ 1 : 1 になる 128 : 1 を基準とし，2048 : 1 から 8 : 1 まで切り替えた．加えて，Small のトラヒックのみ，Large のトラヒックのみを選択する場合についても評価した．その他の設定は基本的に **7.1** と同様である．ただし，図 10 のネットワークを利用し，TCP の ACK などのパケットもトンネリングの対象としている点は **7.1** とは異なる．

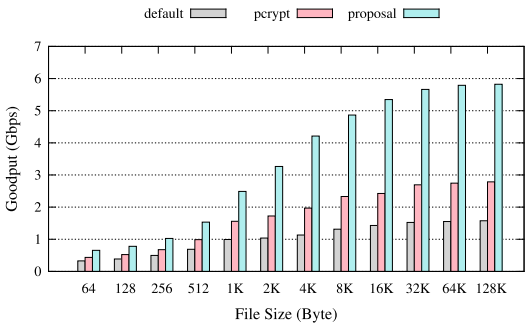

図 11 アプリケーションによるベンチマークにおけるグッドプット

Fig. 11 Goodput of the benchmark using real world applications.

表 2 トラヒックのパターン

Table 2 Patterns of traffic.

名称	メッセージサイズ (Byte)	メッセージ送信回数
Small	64	10
Large	128 K	1

複数のトラヒックが混在するベンチマークにおけるスループットを図 12 に示す．図 12 より，Small のトラヒックのみの場合にはパケット処理によってスループットが低くなり，Large のトラヒックのみの場合にはスループットは高くなった．Small と Large のトラヒックを混在させた場合には，その比率に応じてスループットが変化した．全てのトラヒックの比率において，proposal は default 及び pcrypt と比較して高いスループットを達成した．このことから，複数のトラヒックが混在する実際のクラウド環境においても，提案手法が高いスループットを達成することが期待できる．

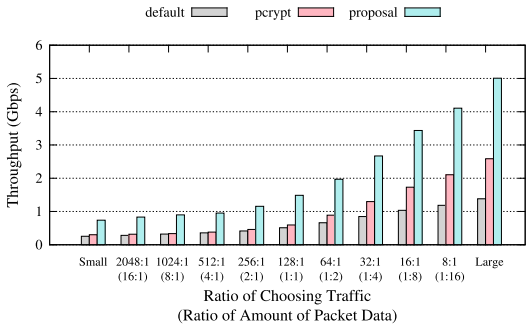

図 12　複数のトラヒックが混在するベンチマークにおけるスループット

Fig. 12　Throughput of the benchmark with multiple patterns.

8. むすび

本論文では，IPsec 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，3 種類の性質の異なるベンチマークによって性能を評価し，提案手法はマルチコア CPU を有効に活用することで，トラヒックの性質にかかわらず既存手法である 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．

文　献

[1] “RFC 4301: Security architecture for the Internet protocol,” http://www.ietf.org/rfc/rfc4301.txt, Dec. 2005.

[2] 小川　拡，齋藤彰一，川島龍太，瀧本栄二，毛利公一，松尾啓志，“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,” 情処学研報インターネットと運用技術，vol.2014, no.10, pp.1–8, June 2014.

[3] “Introduction to Receive Side Scaling,” http://msdn.microsoft.com/en-us/library/windows/hardware/ff556942(v=vs.85).aspx, 参照 Oct. 23, 2014.

[4] “Intel Ethernet Controller X540 Datasheet,” http://www.intel.co.jp/content/dam/www/public/us/en/documents/datasheets/ethernet-x540-datasheet.pdf, March 2014.

[5] S. Klassert, “Parallelizing IPsec: Switching SMP to ‘On’ is not even half the way,” http://www.strongswan.org/docs/Steffen_Klassert_Parallelizing_IPsec.pdf, June 2010.

[6] Y. Liu, D. Xu, W. Song, and Z. Mu, “Design and implementation of high performance IPSec applications with multi-core processors,” 2008 International Seminar on Future Information Technology and Management Engineering, pp.595–598, 2008.

[7] Y. Niu, L. Wu, L. Wang, X. Zhang, and J. Xu, “A configurable IPSec processor for high performance inline security network processor,” 2011 Seventh International Conference on Computational Intelligence and Security, pp.674–678, Dec. 2011.

[8] B. Driessen, T. Guneysu, E.B. Kavun, O. Mischke, C. Paar, and T. Poppelmann, “IPSecco: A lightweight and reconfigurable IPSec core,” 2012 International Conference on Reconfigurable Computing and FPGAs, pp.1–7, Dec. 2012.

[9] V. Ahuja, M. Farrens, and D. Ghosal, “Cache-aware affinitization on commodity multicores for high-speed network flows,” Proc. Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, pp.39–48, 2012.

[10] A. Pesterev, J. Strauss, N. Zeldovich, and R.T. Morris, “Improving network connection locality on multicore systems,” Proc. 7th ACM European Conference on Computer Systems, pp.337–350, 2012.

[11] R. Rosen, Linux Kernel Networking: Implementation and Theory, Apress, 2013.

[12] “nginx news,” http://nginx.org/, 参照 Oct. 23, 2014.

[13] “ab - Apache HTTP server benchmarking tool,” http://httpd.apache.org/docs/2.4/programs/ab.html, 参照 Oct. 23, 2014.

（平成 26 年 10 月 27 日受付，27 年 1 月 28 日再受付）

小川　拡

2013 年豊田工業高等専門学校専攻科情報科学専攻修了．同年名古屋工業大学大学院工学研究科博士前期課程入学，現在に至る．システムソフトウェアに関する研究に従事．

齋藤　彰一

1993 年立命館大学理工学部情報工学科卒業．1995 年同大学大学院博士前期課程修了．1998 年同大学大学院博士後期課程単位習得中退．同年和歌山大学システム工学部情報通信システム学科助手．2003 年同講師，2005 年同助教授．2006 年名古屋工業大学大学院助教授，2007 年同准教授．現在に至る．オペレーティングシステム，インターネット，セキュリティなどの研究に従事．博士（工学），情報処理学会，ACM，IEEE-CS 各会員．

安井　裕亮

2011 年名古屋工業大学工学部情報工学科卒業，2013 年同大学大学院工学研究科博士前期課程修了．同年同大学大学院工学研究科博士後期課程入学，現在に至る．オペレーティングシステムに関する研究に従事．

川島　龍太　（正員）

平成 19 岩手県立大・ソフトウェア情報学研究科博士前期課程了．平成 22 総合研究大学院大学・複合科学研究科了．博士（情報学）．同年株式会社 ACCESS 入社．平成 25 名工大大学院・工学研究科・助教，現在に至る．主に SDN，システムソフトウェアに関する研究に従事．情報処理学会，IEEE 各会員．

瀧本　栄二

1976 年生．1999 年立命館大学理工学部情報学科卒業，2001 年同大学大学院理工学研究科博士前期課程修了，2005 年同研究科博士後期課程単位取得退学，同年（株）ATR 適応コミュニケーション研究所専任研究員，2010 年立命館大学情報理工学部情報システム学科助手，現在に至る．主にシステムソフトウェア，無線通信に関する研究に従事．

毛利　公一　（正員）

1994 年立命館大学理工学部情報工学科卒業，1996 年同大学大学院理工学研究科修士課程情報システム学専攻修了，1999 年同研究科博士課程後期課程総合理工学専攻修了．同年東京農工大学工学部情報コミュニケーション工学科助手，2002 年立命館大学理工学部情報学科講師，2004 年同大学情報理工学部情報システム学科講師，2008 年同准教授，2014 年同教授となり，現在に至る．博士（工学）．オペレーティングシステム，仮想化技術，コンピュータセキュリティ等の研究に従事．情報処理学会，ACM，IEEE-CS，USENIX 各会員．

松尾　啓志　（正員）

昭和 58 名工大・情報卒．昭和 60 同大学大学院修士課程了．同年松下電器産業（株）入社．平 1 名工大大学院博士課程了．同年名工大・電気情報・助手．講師，助教授を経て，平 15 同大大学院教授，現在に至る．分散システム，画像認識，分散協調処理に関する研究に従事．工博．情報処理学会，人工知能学会，IEEE 各会員．