

CPU/FPGA 密結合アーキテクチャを用いた ネットワーク機能仮想化アクセラレーション手法

渡邊義和[†] 柴田誠也[†] 小林悠記[†]
竹中崇[†] 細見岳生[†] 中村 祐一[†]

専用HW装置で従来実現されてきたネットワーク(NW)機能を汎用サーバで代替するNW機能仮想化(NFV)が通信事業者網へ導入されつつあるが、SW処理の性能ボトルネックが懸念されている。また、CPUとFPGAが一チップに統合された密結合アーキテクチャの普及が今後予想される。そこで、本稿ではCPU/FPGA密結合アーキテクチャを用いたネットワーク仮想化アクセラレーション手法を提案する。提案手法は、FPGA活用による性能向上を実現しつつNFVシステムの柔軟性を確保するため、CPUとFPGAの間のインタフェースにNWソフトウェアでよく用いられるDPDK Ring queueを使用する。本稿では、提案手法をXeon+FPGAプラットフォーム試作機上で評価した。その結果、提案手法により、FPGAによるパケット処理に必要なCPU/FPGA間通信を高速かつ効率よく実現できること、また、処理実行HWをCPU/FPGA間で円滑に切り替えられることが確認できた。

Acceleration Method for Network Function Virtualization Using FPGA tightly coupled with CPU

YOSHIKAZU WATANABE[†] SEIYA SHIBATA[†] YUKI KOBAYASHI[†]
TAKASHI TAKENAKA[†] TAKEO HOSOMI[†] YUICHI NAKAMURA[†]

Network function virtualization (NFV) is becoming a new networking architecture for telecom carriers. NFV realizes network functions with software and COTS servers instead of dedicated hardware. While the software-based approach is expected to reduce costs, it could cause performance issues. In near future, FPGA tightly coupled with CPU will get wide spread use and be part of a COTS server. In this paper, we propose an acceleration method for NFV using FPGA tightly coupled with CPU. The method uses DPDK Ring queue, which is often used by network software, as the communication interface between the FPGA and the CPU. It avoids degradation on the flexibility of NFV systems while utilizing the FPGA to accelerate the execution of network functions. We evaluated the method with a prototype of Xeon+FPGA platform. The result shows that the method and FPGA tightly coupled with CPU realize good performance and flexibility for NFV.

1. はじめに

近年、従来専用HW装置で実現されてきたネットワーク(NW)機能を汎用サーバで代替するネットワーク機能仮想化(NFV)[1]が通信事業者網へ導入されつつある。NFVは欧州電気通信標準化機構(ETSI)で標準化検討がなされている。NFVでは、NW機能をソフトウェア(SW)実装し、仮想化技術を用いて汎用サーバ上で実行することでNW機能を実現する。NFVでは、汎用サーバを用いることによる設備コストの削減や、NW機能の柔軟な配置や組み合わせによる提供通信サービスの価値向上等が期待されている。

その一方、NFVのPoC(Proof of Concept)による実証等において、SW実装による性能ボトルネックへの懸念が生じつつある。所望の性能を達成するために多数のCPUが必要となると、準備すべきサーバの台数が増え、設備コスト削減の目的が達成できなくなる。そのため、ETSIではNFVのHWアクセラレーションも並行し検討されている。

別の背景として、クラウドやNFVの実行基盤となるデータセンタへのCPU/FPGA密結合アーキテクチャの普及が挙げられる。Intel社はFPGAを組み込んだデータセンタ向

けプロセッサ(Xeon+FPGA[2])の計画を表明しており、同社は2020年までにクラウドサービスプロバイダの1/3がFPGAを利用すると予想している。Xeon+FPGAは、CPUとFPGAとが一チップに統合されており、メインメモリを共有する。また、FPGAはCPU用キャッシュメモリとコヒーレントなキャッシュ機構を持つ。本稿ではそのようなアーキテクチャをCPU/FPGA密結合アーキテクチャと呼ぶ。IBMおよびXilinxもCPU/FPGA密結合アーキテクチャの実現に向けた提携を発表している。

そこで、本稿ではCPU/FPGA密結合アーキテクチャを用いたネットワーク仮想化アクセラレーション手法を提案する。提案手法は、FPGA活用による性能向上を実現しつつNFVシステムの柔軟性を確保するため、CPUとFPGAの間のインタフェースにNWソフトウェアでよく用いられるDPDK Ring queueを使用する。本稿では、提案手法をXeon+FPGAプラットフォーム試作機上で評価した。その結果、提案手法により、FPGAによるパケット処理に必要なCPU/FPGA間通信を高速かつ効率よく実現できること、また、処理実行HWをCPU/FPGA間で円滑に切り替えられることが確認できた。

[†] 日本電気株式会社 システムプラットフォーム研究所
NEC Corporation. System Platform Research Laboratories.

2. ネットワーク機能仮想化(NFV)とその課題

前述したとおり、NFVでは、NW機能をSWとして実装し仮想化技術を用いて汎用サーバ上で実行する。NW機能の例としては、パケットの転送先を決定するRouting機能、パケットのフィルタリングを行うFirewall機能、パケットの暗号化・復号化を行うIPSec機能、等が挙げられる。

NFVの導入により主に以下の効果が期待されている。

- 効果1：汎用サーバの利用、複数の顧客や複数のサービスによる設備の共用、等による設備コストの削減
- 効果2：NW機能の柔軟な配置や組み合わせによる提供通信サービスの価値向上
- 効果3：SWベースの開発による新規機能・新規サービスの迅速な展開

効果2の例について、図1を用いて説明する。図1では、顧客AにはFirewall機能およびRouting機能を適用する通信サービスを提供し、顧客Bにはそれらに加えIPSec機能を適用する通信サービスを提供しているものとする。NFVでは、NW機能はSWとして実装され、例えば仮想マシン(VM)上で実行される。各顧客のトラフィックは、適用すべき機能を実行しているVMを通過していくように制御される。トラフィック量が低くすべての処理を一台の物理マシンで実行可能な場合、各VMは同じ物理マシンの上に配置される(図1 A)。ここで、例えば顧客Bのトラフィックが増えIPSec機能の処理負荷が高まり一台の物理マシンで処理しきれなくなった場合、物理マシンを追加しIPSec機能のVMを新しい物理マシン上で実行することで対応できる(図1 B)。このように、NFVでは仮想化技術を活用することでNW機能の柔軟な配置や組み合わせを実現し、顧客や通信事業者による様々な機能要件や性能要件に柔軟に対応することができる。

PoCプロジェクト等を通じNFVの実証が進められているが、その中でNW機能のSW実装による性能ボトルネックが課題として顕在化しつつある[3]。所望の性能を達成するために多数のサーバが必要となると、設備コスト削減の効果が得られなくなる。

その解決策としてFPGAやGPGPU等のHWによるアクセラレーションが考えられるが、その活用にあたっては前述した効果を損なわないために以下の課題がある。

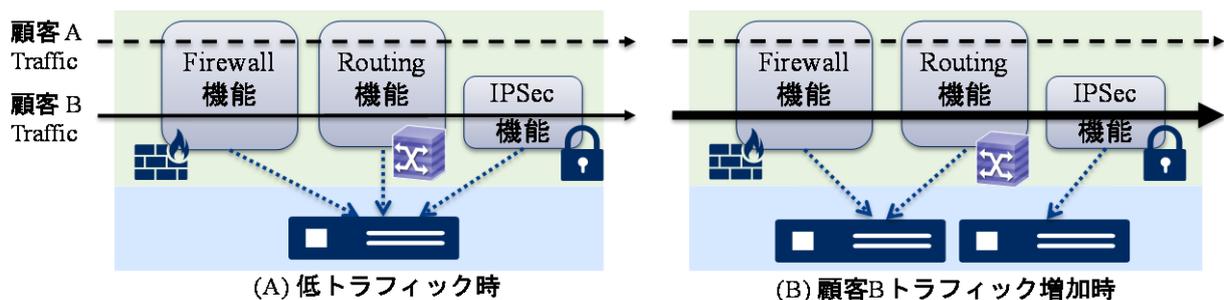


図1 NFVにおけるNW機能の柔軟な配置や組み合わせの例

- 汎用サーバで利用可能であること
- SW実装のNW機能と同様に柔軟な配置が可能であり、かつ他のNW機能と組み合わせ可能であること
- 短い期間や少ないコストで開発可能であること

これらの課題を踏まえ、本稿ではCPU/FPGA密結合アーキテクチャを用いたネットワーク機能仮想化アクセラレーション手法を提案する。

3. 提案するネットワーク機能仮想化アクセラレーション手法

3.1 概観

提案手法は、SW実装のNW機能とFPGA実装のNW機能との間のインターフェースとしてNWソフトウェアでよく用いられるDPDK[4] Ring Queueを使用する。これにより、あるNW機能の配置(CPU or FPGA)を当該NW機能に接続する他のNW機能に影響を与えることなく選択したり変更したりすることが可能となり、柔軟なシステム構成およびアクセラレーションが実現できる。

提案手法は、CPUとFPGAとが一チップに統合されたCPU/FPGA密結合アーキテクチャを活用する。これにより、DPDK Ring QueueによるCPU/FPGA間通信を効率的に実現できる。CPU/FPGA密結合アーキテクチャは将来の汎用サーバで一般的に利用可能となることが期待できるため、提案手法は将来のNFV高速化に有用であると考えられる。

提案手法は、DPDK Ring QueueによるCPU/FPGA間通信処理をカプセル化したモジュール構成をとる。これにより、NW機能をFPGAでアクセラレートする際、そのNW機能に固有の処理のみHW化するだけでよく、短期間かつ低コストに開発することができる。

3.2 DPDKおよびDPDK Ring Queue

NWソフトウェア向けの高速度パケット処理ライブラリとしてDPDK(Data Plane Development Kit)が広く使用されている。DPDKはOSSとして開発されており、CPUでパケットを高速に処理するためのデータ構造や関数群を提供する。

DPDKを使用するNWソフトウェアは、mbuf形式でパケットを記憶し、モジュール間のIFとしてDPDK Ring Queue(DRQ)を使用する。図2にこれらの関係を示す。パケットは、メタデータを格納するmbuf構造体と、パケットデータ自体を格納するパケットバッファとによりメインメモリ

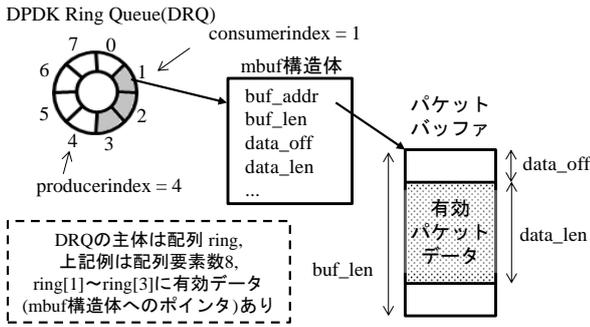


図2 DPDK Ring Queue (DRQ)および mbuf 形式の関係

上に記憶される。mbuf 構造体は、パケットバッファへのポインタ、パケット長、パケットバッファ内での有効データ開始オフセット、等を保持する。DRQ の主体は配列を用いて実現される Queue(FIFO)である。

例えば、パケットを NW 機能 A, NW 機能 B の順に処理すべき場合、両者は DRQ によって接続される。NW 機能 A は処理済みパケットの mbuf 構造体へのポインタ (mbuf ポインタ) を DRQ に enqueue し、NW 機能 B はその DRQ から dequeue した mbuf ポインタを参照してパケット処理を行う。DRQ に enqueue / dequeue するモジュールはそれぞれ producer / consumer と呼ばれ、各モジュールが次に使用するべき配列 index 値が DRQ 管理データとして保持される。例えば、図 2 は producer index が 4 の状態を示しており、次に enqueue が実行される場合、mbuf ポインタは配列 ring[4] に格納され producer index は 5 に更新される。

3.3 SW/FPGA 間インタフェース

提案手法は、SW 実装の NW 機能と FPGA 実装の NW 機能との間のインタフェースとして DRQ を用いる。これにより、FPGA 実装された NW 機能は、他の NW 機能にとって従来の SW 実装された NW 機能と同様に見える。そのため、既存の SW 実装された NW 機能に大きな影響を与えることなく、ある NW 機能を FPGA 実装に置き換えアクセラレーションすることが可能となる。

図 3 に、パケットを NW 機能 A, NW 機能 B の順に処理する場合において、NW 機能 B の実行 HW を CPU から FPGA

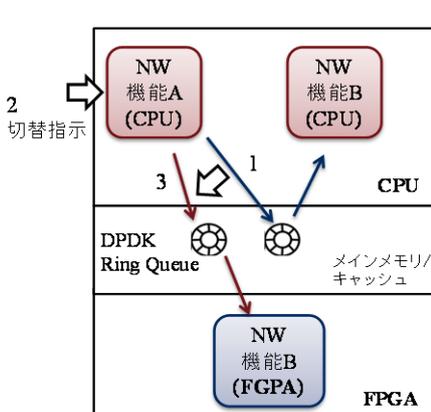


図3 NW 機能 実行 HW 切り替えの例

へ動的に切り替える例を示す。NW 機能 B は CPU および FPGA で実装され動作しており、NW 機能 A からパケットを受け取るための DRQ がそれぞれ用意されている。NW 機能 A は初期状態では処理したパケットを SW 実装された NW 機能 B 用の DRQ へ出力する(図 3-1)。NW 機能 B のアクセラレーションのため FPGA 利用への切り替え指示を受けると、NW 機能 A は出力先 DRQ を FPGA 実装された NW 機能 B 用の DRQ に切り替える(図 3-2)。その後のパケットは FPGA 実装された NW 機能 B で処理される(図 3-3)。このように、提案手法では、NW 機能 A に出力先 DRQ を切り替える機能追加を行うだけで、後続する NW 機能を FPGA によりアクセラレーションでき、さらに NW 機能の実行 HW を動的かつ円滑に切り替えることができる。

3.4 CPU/FPGA 密結合アーキテクチャを用いた CPU/FPGA 間通信の効率化

提案手法は、DRQ による CPU/FPGA 間通信を効率的に実現するため、CPU/FPGA 密結合アーキテクチャを用いる。本稿では、CPU/FPGA 密結合アーキテクチャを以下のように定義する。

- CPU と FPGA が広帯域かつ低遅延なバスで接続され、メインメモリを共有する
- CPU および FPGA はそれぞれキャッシュメモリを備え、両者間のコヒーレンシが保たれる

CPU/FPGA 密結合アーキテクチャを用いることで、FPGA による DRQ 処理を少ない帯域消費および低遅延で実現できる[5]。DRQ では、producer および consumer が相互に index 値を参照して enqueue/dequeue の可否や使用する配列エントリを決定する。そのため、両者は常に最新の index 値を参照する必要がある。CPU/FPGA 密結合アーキテクチャでは、キャッシュメモリ間にコヒーレンシがあるため、CPU および FPGA は index 値をキャッシュメモリに格納し参照および更新することが可能である。そのため、DRQ 処理を少ない帯域消費および低遅延で実現できる。一方、非 CPU/FPGA 密結合アーキテクチャの場合、CPU/FPGA 間で DRQ を共有できない、または、常にメインメモリにアクセ

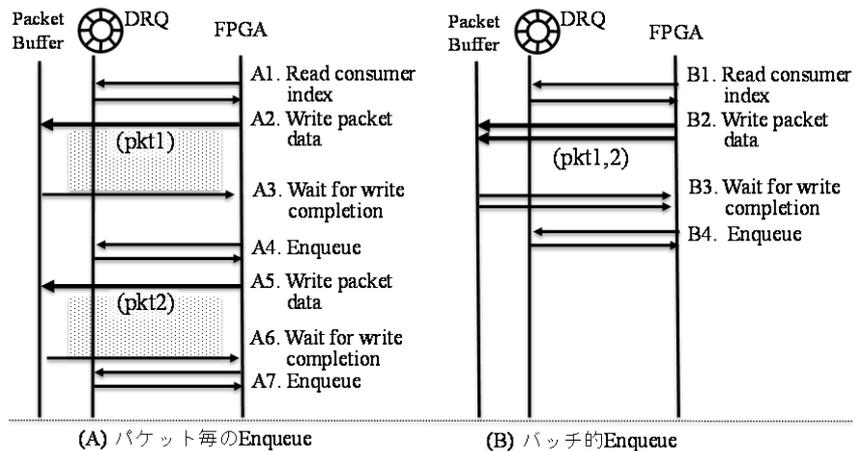


図4 バッチ的な Enqueue の動作

スする形で使用する必要があるためオーバーヘッド(帯域・遅延)が大きいという問題が生じる。

提案手法では、FPGA が処理済み後のパケットデータをメモリに書き込み DRQ に enqueue する際、複数パケットをバッチ的に enqueue し高いバス利用効率を実現する。データの整合性を保つため、DRQ への enqueue(mbuf ポインタの書き込みおよび producer index の更新)は、そのパケットデータのメモリ書き込みが完了した後に行う必要がある(図 4 A4, A7)。さもなくば、consumer が mbuf ポインタを dequeue しパケットデータを参照した際、producer による更新前の状態のパケットデータを参照する恐れがある。しかし、パケットデータのメモリ書き込みを待っている間(図 4A ハッチ部分)はバスをほとんど利用しないことになるため、バス利用効率が低下し処理性能が劣化する。そこで、提案手法では、パケット毎に DRQ へ enqueue するのではなく、複数のパケットデータをバッチ的にメモリに書き込み(図 4 B2)、それらの完了後に一括して DRQ に enqueue する(図 4 B4)。これにより、バス利用効率の向上が期待できる。ただし、通信遅延が増加することになるため、適切なバッチサイズを選択することが必要である。

3.5 FPGA モジュール構成

図 5 に提案手法の FPGA モジュール構成を示す。DRQ アクセス回路は FPGA 実装の NW 機能に共通的に必要となるため、ライブラリ化し再利用可能とする。一方、NW 機能回路は NW 機能特有の処理を実装する回路である。このような構成により、ある NW 機能をアクセラレートするときはその NW 機能回路のみ実装するだけでよく、NW 機能のアクセラレーションを短期間かつ低コストに実現できる。

Configuration 回路は、ring_dequeue および ring_enqueue に対し DRQ の情報として DRQ の識別子(DRQ ID)と DRQ へのポインタとの組のリストを設定する。また、Configuration 回路は、packet_read に対しパケットデータのどの部分を読み込むべきかの情報(オフセットおよびサイズ)を設定する。例えば、NW 機能回路がヘッダを見てパケットを振り分ける NW 機能を実装している場合、ヘッダ以外のパケットデータを読み込む必要はない。その場合、オフセット=0、サイズ=ヘッダ長、とすることでヘッダ以外

の読み込みを抑制しバス帯域消費を削減することができる。

Dequeue 用 DRQ アクセス回路である ring_dequeue は DRQ の consumer として動作し、mbuf ポインタを読み込み mbuf_read に供給する。その際、DRQ の識別子(DRQ ID)と mbuf ポインタの組を渡す(FIFO1)。mbuf_read は mbuf ポインタを基に mbuf 構造体を読み込み、packet_read に供給する(FIFO2)。packet_read は mbuf 構造体の情報を基にパケットバッファからパケットデータを読み込み NW 機能回路に供給する(FIFO3=DRQ/mbuf 情報, FIFO4=パケットデータ)。Enqueue 用 DRQ アクセス回路である ring_enqueue は、パケットデータおよび mbuf 構造体のメモリへの書き込み、および DRQ への mbuf ポインタの enqueue を行う。

4. 実装および評価

4.1 実装および評価環境

提案手法を Xeon+FPGA 評価機上に実装し評価した。DRQ アクセス回路および NW 機能回路の実装には C 言語ベースの高位合成ツール CyberWorkBench を用いた。使用した評価機は Broadwell 世代の CPU と Arria10 FPGA とを統合した Xeon+FPGA を搭載しており、CPU と FPGA との間が QPI および PCI Express で接続されている。

4.2 評価

最初に、CPU/FPGA 間の DRQ による通信の評価を行う。FPGA での高い処理スループットを実現するには、処理に必要なデータすなわちパケットを DRQ 経由で CPU との間で高速かつ効率よく通信できる必要がある。次節では、提案手法により、十分な通信性能が達成できること、および低オーバーヘッドによる通信を実現できることを示す。

次に、NW 機能のアクセラレーション効果を評価する。本稿では、パケット暗号化を NW 機能の例にとり、FPGA により処理性能を向上させられること、および NW 機能の実行 HW を柔軟かつ円滑に切り替えられること、を示す。

本稿では達成すべき通信性能の基準として 40Gbps を用いる。この値は、NFV の対象となる通信処理基幹装置への 40GbE IF の普及が近年進みつつあることから設定した[6]。

4.2.1 CPU/FPGA 間の DRQ による通信の評価

CPU/FPGA 間の DRQ に 40Gbps のトラフィックを入力し、

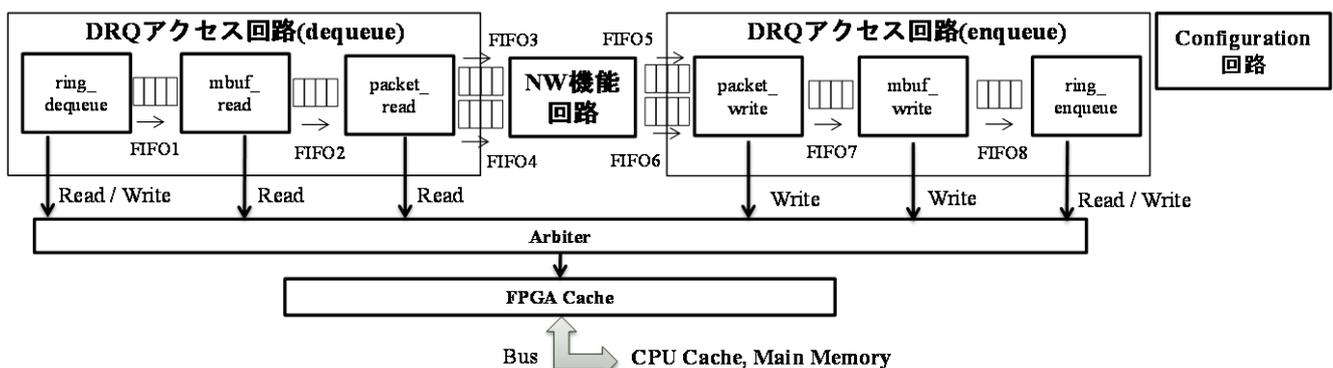


図 5 提案手法の FPGA モジュール構成

FPGA による dequeue および enqueue 性能を評価した。複数のパケットサイズおよび複数のバッチサイズ(パケット数)について評価した。対向の CPU 処理がボトルネックとならないよう、十分な数の CPU コアを使用した。なお、本評価の結果は、使用した評価機の性能限界を示すものではない。

FPGA による dequeue 性能(CPU→FPGA の通信性能)の評価結果を図 6 に示す。図 6 は FPGA が dequeue できたパケットデータのスループットを示している。本評価におけるバッチ的 dequeue は、バッチサイズ分の数の mbuf ポインタ読み込みが完了するごとに consumer index を書き込むこととした。評価の結果、パケットサイズ 128 バイト以上の場合に入力 40Gbps 全てを FPGA は受信することができた。バッチサイズによる影響はほとんど見られなかった。これは、dequeue する際、CPU による producer index の更新が終わると FPGA は Ring を読み進めることができ、consumer index の更新処理が性能劣化要因になりにくいためである。なお、パケットサイズが 64 の場合に 40Gbps を達成できなかった。これは、小さいパケットは高頻度の処理(64B 時で約 58M 回/s)が必要であり、今回我々が作成した回路の最適化がそれらすべてを処理するのに不十分であったためである(enqueue における 128B 以下の場合も同様)。この点は今後の課題としたい。図 7 に Ring dequeue 処理に消費したバス帯域を示す。バッチサイズが大きくなるにつれて index 更新間隔が減ることから、消費バス帯域が減る傾向にある。

FPGA による enqueue 性能(FPGA→CPU の通信性能)の評価結果を図 8 に示す。図 8 は FPGA が enqueue できたパケットデータのスループットを示している。バッチサイズ 64 以上、パケットサイズ 256 バイト以上の場合に入力 40Gbps 全てを FPGA は送信することができた。図 9 に Ring enqueue 処理に消費したバス帯域を示す。バッチサイズが小さい場

合、高頻度に index 更新が行われ、多くの帯域を消費している。バッチサイズを 32 以上にすることで消費帯域を大きく減らせることが分かった。

以上のことから、40Gbps の通信を実現し、かつ DRQ による帯域オーバーヘッドを少なくするためにはバッチサイズを 64 以上にすればよいことが分かった。一方、バッチサイズを大きくすると通信遅延が大きくなることが予想される。今回、遅延に関し詳細に評価できておらず、この点は今後の課題としたい。

4.2.2 アクセラレーション効果の評価

NW 機能としてパケット暗号化を SW および FPGA で実装し、処理性能を比較した。暗号化方式には、AES-128CBC を用い、SW 実装での暗号化処理には OpenSSL (CPU 拡張機能 AES-NI 利用)を使用した。バッチサイズには、SW/FPGA とも 64 を使用した。パケットサイズには 1514 バイトを使用し、入力レートを 40Gbps (3.3Mpps)とした。

図 10 に、40Gbps の入力トラフィックを与えた場合の出力スループットを示す。CPU で処理した場合については、一 CPU コアのみを使用した結果と、複数コアを使用した結果を示している。図 10 より、提案手法を用いることでパケット暗号化 NW 機能を少なくとも約 8 倍アクセラレート可能なことが分かる(対一 CPU コア)。

次に、NW 機能を実行する HW の切り替えについて評価する。初期状態において入力トラフィック(40Gbps)の暗号化処理を CPU で実行しておき、図 3 に示した方法により FPGA による処理に切り替えた。本評価において、図 3 の NW 機能 A はトラフィック生成機能、NW 機能 B はパケット暗号化機能である。また、NW 機能 B の出力を受け取る出力測定 NW 機能(SW 実装)も使用した。FPGA による処理に切り替えた後、再び CPU による処理を行うよう切り替え

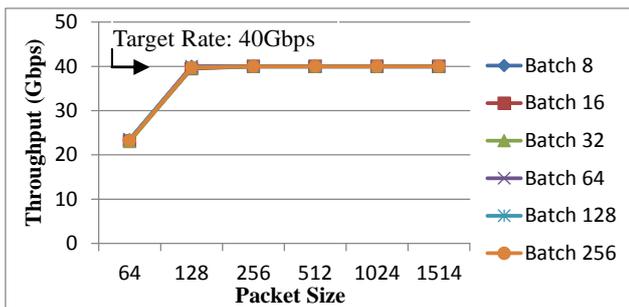


図 6 Dequeue スループット評価結果

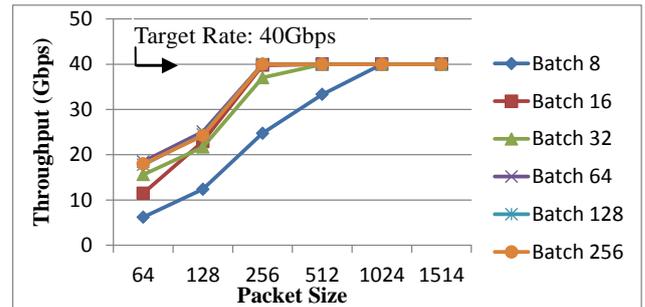


図 8 Enqueue スループット評価結果

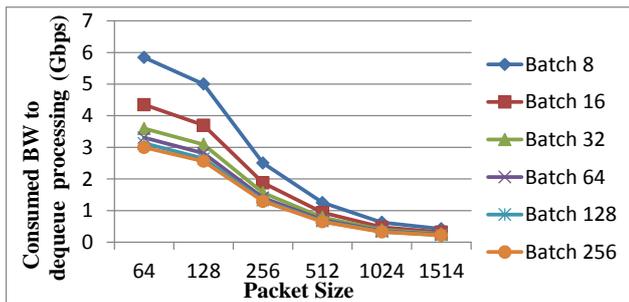


図 7 Dequeue オーバヘッド評価結果

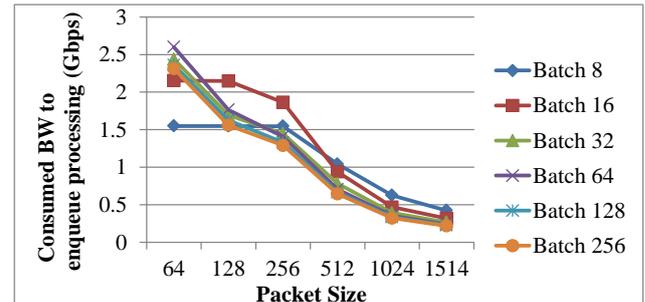


図 9 Enqueue オーバヘッド評価結果

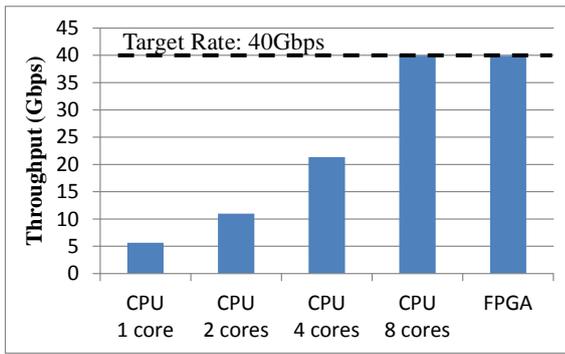


図 10 パケット暗号化処理スループット評価結果

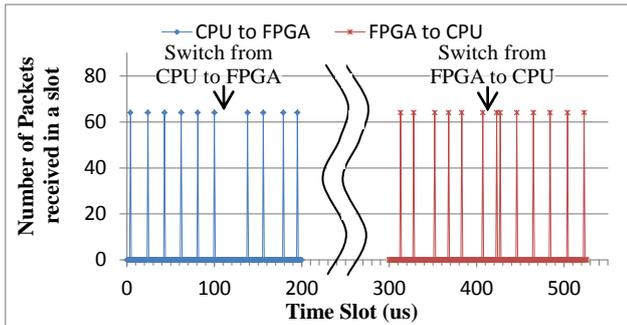


図 11 処理実行 HW 切り替え時のパケット受信数(us 毎)

た。これら一連の動作をさせた際の、出力スループットおよびパケットロス数を出力測定 NW 機能で測定した。

切り替え実施時を含め、秒単位で算出した出力スループットに特異な変動は見られなかった。また、パケットロス数は常に 0 であった。図 11 に、切り替え実行時付近におけるマイクロ秒毎の受信パケット数(出力測定 NW 機能で測定)を示す。暗号化処理実行 HW を CPU から FPGA に切り替えた際、パケットが受信されない空間時間が 20us 前後生じた。この値は、パケット発生間隔(3.3Mpps の場合約 300ns 秒)にバッチサイズを乗算した値と概ね合致する。バッチ処理のためのパケット処理待ちや FPGA による通信および処理遅延が上記空白時間の原因と考えられる。一方、FPGA から CPU への切り替えの際には、FPGA からの出力パケット全てが受信された後、速やかに CPU からの出力パケットが受信された。これは、FPGA で処理される最後のパケットが出力されるまでの間に、新たに CPU で処理され始めるパケットの暗号化処理が完了していたためと考えられる。

以上のことから、提案手法により NW 機能を FPGA でアクセラレートできるとともに、実行 HW を円滑に切り替え可能なことが確認できた。

5. 関連研究

ReClick[7]は、ソフトウェア NW 機能フレームワークである Click 向けの FPGA 活用フレームワークである。Click では NW 機能は細かくモジュール化され、複数のモジュールのインスタンスを生成・接続することでより複雑な NW 機能を実現できるようになっている。また、Click では高位言語による記述も可能である。ReClick は、Click モジュー

ルを FPGA で置換できるようにしており、また、高位言語の利用も考慮されている。ReClick を用いる場合、Click および ReClick の規定(IF, データ構造等)に合わせて NW 機能を実装する必要がある。Click は学术界では多く用いられているものの、実運用環境での利用は限られており、特に NFV へ適用する場合には NW 機能の再実装が必要となる。一方、提案手法は実運用向けネットワーク SW でよく用いられる DPDK を使用しており、既存の SW に大きな変更を加えることなく、FPGA を活用することが可能である。また、NW 機能の FPGA 実装に高位合成ツールを活用することで、開発効率の向上が期待される。

Lagopus FPGA[8]は、DPDK を使用し実装された NW スイッチソフトウェア Lagopus のパケット分類 NW 機能を NIC 上の FPGA にオフロードしスイッチング性能向上および電力消費削減に成功している。パケット分類 NW 機能やパケット暗号化 NW 機能をはじめ、NFV で用いられる様々な NW 機能が FPGA で高速化可能であると考えられる。それらの NW 機能に提案手法を適用することで、柔軟かつ高性能な NFV システムを実現できる。

6. おわりに

本稿では、CPU/FPGA 密結合アーキテクチャを用いたネットワーク仮想化アクセラレーション手法を提案した。また、提案手法を Xeon+FPGA プラットフォーム試作機上で評価し報告した。今後の課題として、通信および処理遅延の詳細評価およびその改善、小サイズパケット処理時の効率化、高位合成ツール等を用いた NW 機能回路の開発容易化が挙げられる。

謝辞 本稿における実装および評価は Intel 社のご協力の下で行われた。ここに謹んで御礼申し上げる。

参考文献

- [1] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. De Turck and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," in IEEE Communications Surveys & Tutorials, vol. 18, no. 1, pp. 236-262, Firstquarter 2016.
- [2] Gupta, Prabhat K. "Xeon+ FPGA Platform for the Data Center." Fourth Workshop on the Intersections of Computer Architecture and Reconfigurable Logic. Vol. 119. 2015.
- [3] ETSI GS NFV-IFA 001 V1.1.1 (2015-12), "(NFV); Acceleration Technologies; Report on Acceleration Technologies & Use Cases"
- [4] DPDK Data Plane Development Kit, <http://dpdk.org>
- [5] Young-kyu Choi, et al., "A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms," Proceedings of the 53rd Design Automation Conference (DAC), Austin, TX, June 2016.
- [6] "Huawei eLTE products". http://enterprise.huawei.com/ilink/cnenterprise/download/HW_328207
- [7] D. Unnikrishnan, J. Lu, L. Gao and R. Tessier, "ReClick - A Modular Dataplane Design Framework for FPGA-Based Network Virtualization," ANCS 2011 Seventh ACM/IEEE Symposium on, 2011.
- [8] K. Yamazaki, et al., "Lagopus FPGA- a reprogrammable data plane for high-performance software SDN switches," Hot Chips 27. 2015.