Intel DPDKを用いたネットワーク遅延エミュレータの開発

明田 修平[1,a)]　広渕 崇宏[2]　瀧本 栄二[1]　毛利 公一[1]　高野 了成[2]

概要：オペレーティングシステムのネットワークスタックをベースにしたソフトウェア実装によるネットワークエミュレータでは，高パケット送信レートに対して伝送遅延を再現するのはゆらぎが大きくなり困難である．そこで，Intel DPDK を用いたネットワーク遅延エミュレータを開発している．DPDK は，広帯域向けのパケット処理フレームワークであり，パケット処理遅延が小さい．DPDK を用いることで，高パケット送信レートに対しても伝送遅延のゆらぎを抑え，高精度なエミュレーションを可能とした．IP パケット長 1500B の 10Gbps で評価した結果，netem と比較してゆらぎの指標である標準偏差を 97 %改善した．また，ショートパケットでは，netem ではパケット処理が追いつかず評価できなかったが，開発したエミュレータではパケット長 1500B と同程度の精度で伝送遅延をエミュレーションできることを確認した．

キーワード：ネットワークエミュレータ，Intel DPDK，10Gb Ethernet，パケット処理

1. はじめに

ネットワークは，我々の生活に欠かせない技術である．世界中と接続できるインターネットや家庭内のパソコンや IoT デバイスを含めたローカルネットワーク，クラウドサービスを提供するためのサーバ間データ通信などと様々である．ネットワークリンクはそれぞれ異なる可用帯域や遅延を有し，それらの性能特性がアプリケーションの性能に大きな影響を与える．研究開発の現場においては，対象とするネットワークの特性を再現する（エミュレーションする）技術が必要とされてきた．

異なる特性のネットワークリンクを仮想的に再現するソフトウェアとして，Linux カーネルに標準搭載されている netem[1]，FreeBSD の dummynet[2] 等が広く使われている．しかし，オペレーティングシステムに搭載されている従来のネットワークスタックを拡張して実装されたネットワークエミュレータは，今日普及しつつある広帯域のネットワーク（e.g., 10/40/100 GbE）の処理速度に対応することが難しい [3]．高速なトラフィック（主にショートパケット）に対してパケット処理が間に合わず大量のパケットロスが生じてしまう．また，仮にパケット処理が間に合ったとしても，転送処理に要する時間のゆらぎ（ジッタ）が大きく，マイクロ秒オーダの精度でネットワーク遅延を再現することは困難である [4]．一例を挙げると，多数のノード間において同期を取りながら計算と通信を進めるハイパフォーマンスコンピューティングのアプリケーションにおいては，ネットワークの伝送遅延が，大きく性能を左右しうる場合がある．アプリケーションの性能をエミュレーション環境で議論するためには，10Gb Ethernet や Infiniband，Myrinet 等の広帯域かつ低遅延のネットワークを再現する必要がある．これらのネットワークでは，マイクロ秒オーダでの制御が必要となり [5]，netem を用いたネットワークエミュレーションでは要求を満たせない．これまで広帯域かつ高精度なネットワークのエミュレーションは，FPGA などハードウェアを用いて行われてきた．しかし，専用のハードウェアを用いる場合，デバイスが高価であり，かつ拡張やメンテナンスに専門の技術者が必要であり開発コストが大きくなる．

そこで，我々は，Intel DPDK[6]（以下，DPDK とする）を用いたネットワークエミュレータである DEMU を開発している．DPDK は，160Mpps（packet per second）のトラフィックを汎用サーバ上でパケット処理可能であるとうたわれている．DPDK は，各スレッドが CPU コアを 1 つ占有することで，無駄なコンテキストスイッチを防ぎ，メモリをラージページ（hugetlbfs）を用いて管理することで TLB ミスを緩和するなどの工夫により従来の Linux ネットワークスタックよりも高速かつ低遅延なパケット処理を実現する．DPDK は，ネットワークの処理で必要となる機能をライブラリ化しており，それらを組み合わせてパケット処理プログラムをユーザアプリケーションとして実装できる．今回開発したネットワークエミュレータは，ブリッジ

1　立命館大学
2　国立研究開発法人 産業技術総合研究所
a)　saketa@asl.cs.ritsumei.ac.jp

© 2016 Information Processing Society of Japan

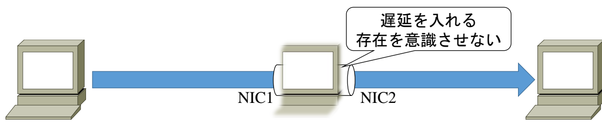

図 1 DEMU のコンセプト

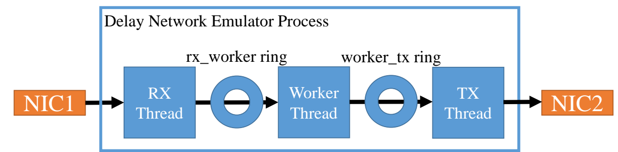

図 2 全体構成

として動作し，あるポートから受信したパケットに対して，内部で目的の伝送遅延を再現して別のポートから送出する．挿入する遅延時間は，マイクロ秒オーダで指定可能とする．本論文の貢献は，ソフトウェア実装による，10GbE でのショートパケットに対しても，ジッタの小さい高精度な伝送遅延のエミュレーションを可能とする．Linux をベースにしたソフトウェア実装のエミュレータで，どこまで正確にネットワークをエミュレーションできるのかに挑む．

以下，本論文では，2 章で今回開発したネットワーク遅延エミュレータ DEMU について述べ，3 章で DEMU と netem における伝送遅延のエミュレーション精度の評価について述べる．4 章で関連研究について述べ，5 章で本論文をまとめる．

2. DEMU

2.1 概要

今回作成した DEMU は，**図 1** に示すように，2 つの NIC をそれぞれ両端のノードと接続し，ブリッジとして実現する．遅延のエミュレーションは，一方の NIC から受信したパケットを指定した時間経過後にもう一方の NIC に送信することで実現する．パケットを中継する際，パケットの参照及び改変は行わない．すなわち，他のノードから DEMU の存在を意識させない．また，指定する遅延時刻はマイクロ秒単位とする．DEMU は，DPDK アプリケーションとして実装し，Linux カーネルのプロトコルスタックを経由せず，NIC から直接アプリケーションへパケットが転送される．

DEMU は，今後，パケットロスや帯域制限などもエミュレーションすることを目指し，そのベース作りを担う．なお，本論文では，解説および評価を簡易にするために，一方向通信にのみ限定して述べる．

2.2 設計

DEMU は，**図 2** に示すように，受信スレッド，遅延をエミュレーションするワーカスレッド，送信スレッドの 3 つのスレッドから構成される．スレッド間のパケット転送は，DPDK ライブラリの rte_ring を用いたリングバッファを経由して行う．なお，開発当初は，送信，遅延のエミュレーション処理，受信処理をすべて単一のスレッド内で行う設計としていた．しかし，10GbE 相当のトラフィックにおいて受信処理が間に合わず多くのパケットロスが生じたため，各処理を別々のスレッドで行う設計とした．NIC の受信用リングバッファからの取りこぼしを防ぎ，また送信用リングバッファのデータ枯渇を避ける効果がある．

2.2.1 各スレッドの設計

受信スレッドでは，DPDK の受信 API である rte_eth_rx_burst() を用いて最大 32 パケットまとめてパケットを受信する．なお，受信処理は，割込みは用いずポーリング処理で行う．そのため，受信スレッドは，ビジーループでこの受信 API を呼出し続ける．受信したパケットは，遅延エミュレーションのために，1 パケットごとに受信した時刻を記録する．この遅延処理に関しては次節にて詳細を述べる．受信したパケットは，rte_ring_sp_enqueue_burst() を用いて rx_worker リングバッファへエンキューする．このとき，リングバッファが満杯でありエンキューできなかった場合は，伝送遅延の増加を抑えるために，エンキューできなかったパケットをメモリ解放し破棄する．

ワーカスレッドでは，rx_worker リングバッファを rte_ring_sc_dequeue_burst() を用いてビジーループで参照し続け，パケットが存在した場合最大 32 パケットまとめて取り出す．遅延処理後，送信するパケットは rte_ring_sp_enqueue() を用いて 1 パケットごとに worker_tx リングバッファへエンキューする．

送信スレッドでは，worker_tx リングバッファからパケットをデキューしそのまま NIC へ送信する．worker_tx リングバッファには，遅延処理されたパケットがエンキューされている．そのため，迅速に worker_tx リングバッファからデキューし送信する必要がある．送信スレッドでは，ビジーループで worker_tx リングバッファを rte_ring_sc_dequeue_burst() を用いて参照し続け，パケットが存在した場合最大 32 パケットまとめてエンキューする．エンキューしたパケットは，すぐさま rte_eth_tx_burst() を用いて最大 32 パケットまとめて送信する．なお，NIC が持つリングバッファが満杯で処理できなかったパケットに関しては，伝送遅延の増加を抑えるためにメモリ解放し破棄する．

2.2.2 遅延処理の設計

パケットを指定時間遅延させるためには，パケットごとに受信時刻を管理および送信のタイミングを制御する必要がある．DPDK においてパケットは，rte_mbuf 構造体で管理されている．rte_mbuf は，Linux の sk_buff 構造体のように，パケットの実体および管理データを保持する構造体である．rte_mbuf 構造体には，ユーザが自由に利用可能な udata64 メンバ（uint64_t 型）が存在する．受信時刻は，

© 2016 Information Processing Society of Japan

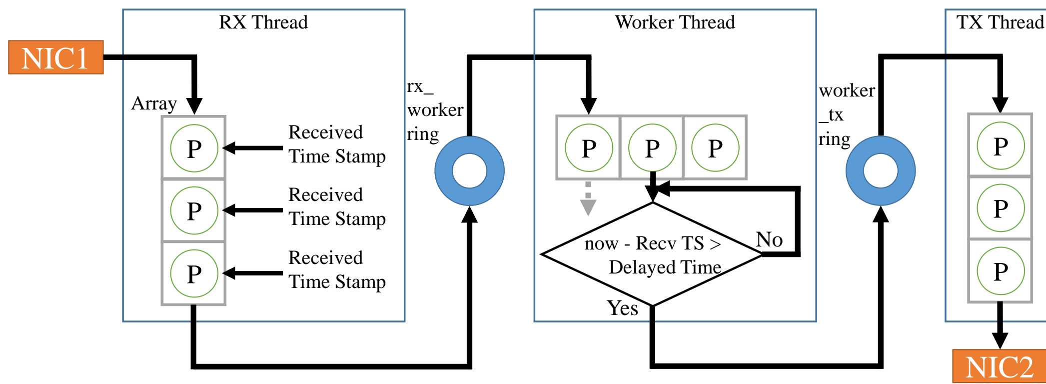

図 3 遅延処理

この udata64 メンバに rdtsc 命令により取得したタイムスタンプをパケットごとに記録する．

図 3 は，各スレッドでの遅延をエミュレーションするための処理手順を示している．受信スレッドでは，バースト的に受信したすべてのパケットに対してタイムスタンプを付与する．送信のタイミング（遅延の挿入）の制御は，ワーカスレッドで行い，1 パケットごとに現在時刻と udata64 メンバ内の受信時刻から行う．現在時刻と受信時刻との差が挿入する遅延時間を超えた場合，worker_tx リングバッファへエンキューする．また，遅延時間の指定方法は，プロセス実行時の引数で指定する．

2.2.3 パケットの管理

DPDK では，hugetlbfs を用いて物理メモリをユーザ空間へマップし，この領域にパケットを保存することで TLB ミスの発生を抑制している．DEMU では，初期化時にこのメモリ空間から利用する領域を rte_pktmbuf_pool_create() を用いて確保し，この領域にパケットを保存する．rte_pktmbuf_pool_create() は，rte_mempool ライブラリとして，何パケット分の領域を確保するか，1 パケットあたりのサイズを指定する．DEMU は，パケット受信時に rte_mempool へパケットを保存し，以降プロセス内ではポインタを用いてリングバッファを経由してスレッド間を転送する．そのため，パケットデータのコピーは発生しない．

rte_mempool のサイズは，rx_worker と worker_tx の 2 つのリングバッファのサイズを指定する．プロトタイプ実装では，10Gbps で IP パケット長が 46B のトラフィックに対して 1 秒間待機可能とするため 15M パケット分の領域が必要である．そのため，各リングバッファのサイズは，rx_worker で 2^{24} (16,777,216) パケット分とし，worker_tx は 512 パケット分とした．なお，rte_ring のサイズは 2 のべき乗にする必要がある．また，1 パケットあたりのサイズは，DPDK ライブラリのデフォルト値である 2,176B とした．

2.3 スレッドの管理

DEMU は，3 つのスレッドを用いてパケットを処理する．これら 3 つのスレッドは，CPU を占有し続ける必要がある．DPDK では，プロセス，スレッドに対して CPU Affinity を設定することで，プロセス，スレッドがコア間を遷移しないようにする．これにより，プロセス，スレッドは，コンテキストスイッチの影響を受けずにビジーループでそれぞれ割り当てられた処理を行う．しかし，Linux のプロセススケジューラは，別プロセスを DPDK プロセスが占有したい CPU に割り当てることもある．そのため，DPDK プロセスは，CPU を占有できずにコンテキストスイッチの影響を受けることがある．これにより，遅延エミュレーションの精度が低下することを確認している．

そこで，DEMU を動作させる Linux 起動時の boot オプションに，isolcpus を付与する．DEMU では 3 つのスレッドを用いるため，isolcpus で 3 コア分指定し Linux のプロセススケジューラの対象から除外する．また，DPDK プロセスは，3 つのコアにそれぞれスレッドを配置するように CPU Affinity を設定する．これにより，Linux のスケジューラは指定されたコアにプロセスを割り当てなくなり，DEMU の 3 つのスレッドは CPU を占有することができ，遅延エミュレーションの精度が向上する．

3. 性能評価

開発した DEMU の精度を確認するために，Linux を用いて性能評価を行った．また，比較対象として Linux にデフォルトで搭載されている netem[1] でも同様の評価を行った．

3.1 評価環境

評価で用いたネットワークの構成は**図 4** とし，パケットの生成および遅延時間の計測には産業技術総合研究所が開発したネットワークテストベッド GtrcNET-10p3[7] を用いた．GtrcNET-10p3 は，大規模 FPGA と 10GbE I/F および DRAM を各 3 ポート接続した構成である．パケッ

© 2016 Information Processing Society of Japan

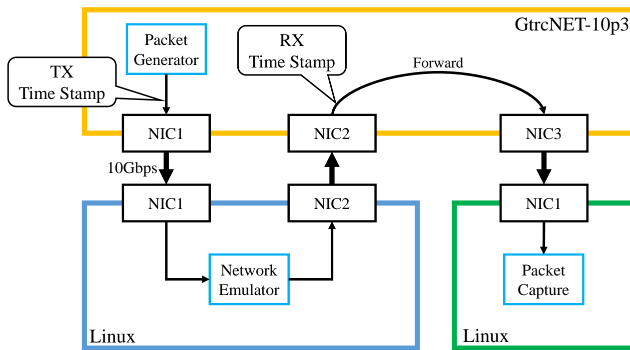

図 4 ネットワーク構成

表 1 Linux マシン環境

Processor	Xeon E5-2430L 2.00GHz × 2
Mother Board	Supermicro X9DBL-iF
Memory	DDR3-1333 8GB × 6
NIC	Intel X520-DA2
OS	Ubuntu 16.04.1 (Kernel 4.4.0-42)

ト送信時にパケットのペイロードに RFC 1305 に準拠したタイムスタンプを付与して送信し，パケット受信時にも同様にタイムスタンプを付与する．この受信タイムスタンプと送信タイムスタンプとの差から遅延時間を計測する．なお，GtrcNET-10p3 の分解能は 25.6 ns である．本評価では 10 ns 以下の位は切り捨てる．DEMU および netem を動作させる Linux マシンは，**表 1** の構成であり，GtrcNET-10p3 と 2 つの NIC で接続する．また，GtrcNET-10p3 が受信したパケットは，タイムスタンプを付与後に別ポートから送信し，もう一台の Linux マシンでそのパケットのキャプチャを行う．

3.2 DEMU の評価

まず，今回作成した DEMU を用いて，遅延エミュレーションの精度を評価した．本評価では，IP パケット長が 1500B と 46B の UDP パケットで行った．パケット長それぞれの設定において，約 10 秒間，ほぼ 10GbE の限界帯域幅の送信トラフィックを生成した（パケット長 1500B で 9.743 Gbps，パケット長 46B で 6.522 Gbps）．なお，GtrcNET-10p3 では 10GbE の論理限界値そのものの送信レート（それぞれ 9.844 Gbps と 7.143 Gbps）でトラフィックを生成することができるが，この送信レートでは Linux の NIC での処理が追いつかず遅延時間が線形増加する現象が見られた．そのため，本評価では，送信レートを上述のように論理限界値から若干低下させている．挿入した遅延時間は，0 s（遅延挿入なし），1 µs，10 µs，100 µs，1 ms，10 ms，100 ms，1 s の 8 パターンとした．これらのパターンで遅延時間を計測し，遅延時間の最大値，最小値，平均値，標準偏差，再現誤差を算出した．再現誤差は，遅延挿入なしでの平均値と挿入した遅延時間との和と，それぞれエミュレーション結果の平均値との誤差である．すなわち，再現誤差が 0 に近いほど，遅延エミュレーションの精度が高いことを表す．

表 2 がパケット長が 1500B のときの，**表 3** が 46B のときの評価結果である．標準偏差を見ると，1500B で 2.5 µs 前後であり，46B で 2.0 から 3.4 µs とジッタは小さかった．また，再現誤差を見ると，1 s 間の遅延挿入を除くと，1500B で 0.7 µs 以下，46B で 1.7 µs 以下と遅延のエミュレーション精度も高かった．1 s 間の遅延挿入で再現誤差が大きくなった理由として，大容量のメモリを消費することからメモリへのアクセスに対するパフォーマンス低下が要因として考えられる [8]．さらに，パケット長が 1500B と 46B とで，遅延時間に大きな差は見られなかった．そのため，実運用するにあたり，大小様々なパケットに対しても，同様に伝送遅延をエミュレーション可能である．

3.3 netem の評価結果

Linux にデフォルトで搭載されている netem を用いて，前節と同様の評価を行った．Linux では，Bridge ではなく IP forwarding で NIC 間を接続した．これは，Linux カーネルのソースコードにおいて，ルーティング処理の方が高度に最適化されているため，ブリッジ処理よりも高いスループットを出すことができることによる [3]．また，評価実験環境において，Linux netem は，IP パケット長 46B とショートパケットに対して 10GbE の論理限界値のスループットを出すことができなかった．そのため，IP パケット長 1500B では 9.743 Gbps と 1 Gbps，IP パケット長 46B は 1 Gbps の送信レートで約 10 秒間トラフィックを生成して評価した．また，netem と Linux の設定において，キューサイズをそれぞれ 1 秒間待機可能なサイズを設定した．

表 4 が 10Gbps でパケット長が 1500B のときの，**表 5** が 1Gbps でパケット長が 1500B のときの，**表 6** が 46B のときの評価結果である．10Gbps のトラフィックでは，標準偏差が 107 から 431 µs であり，再現誤差も 1 ms 前後と DEMU と比較してもジッタが大きく遅延エミュレーションの精度も悪かった．1Gbps のトラフィックに関しては，パケット長が 1500B では標準偏差が 20 µs 前後であり，再現誤差も 1 から 11.7 µs であった．一方，パケット長が 46B では，標準偏差が 3 ms 前後と大きく，再現誤差も 3.5 から 5.4 ms と大きかった．ショートパケットのトラフィックでは，1Gbps でも Linux の処理が追いついておらず伝送遅延のエミュレーション精度が悪かった．

3.4 考察

今回開発した DPDK を用いた DEMU は，netem と比較しても高い遅延エミュレーションの精度および遅延のジッタが小さい特徴を有していることを確認した．10Gbps，IP パケット長 1500B のトラフィックにおいて DEMU と netem とを比較すると，遅延挿入なしでの標準偏差は DEMU が

© 2016 Information Processing Society of Japan

表 2 10Gbps，パケット長 1500B での DEMU の遅延時間精度（単位：µs）

挿入した遅延時間	0	1	10	100	1,000	10,000	100,000	1,000,000
最大値	31.6	31.7	43.2	131.2	1,031.4	10,030.4	100,031.9	1,000,044.9
最小値	9.1	9.8	18.9	108.8	1,008.9	10,008.8	100,009.7	1,000,017.8
平均値	12.1	12.6	21.5	111.6	1,011.8	10,011.6	100,012.8	1,000,021.2
標準偏差	2.5	2.4	2.3	2.3	2.4	2.5	2.6	2.6
再現誤差	—	0.5	0.6	0.5	0.3	0.5	0.7	9.1

再現誤差：遅延挿入なしでの平均値と挿入した遅延時間との和と，平均値との誤差

表 3 10Gbps，パケット長 46B での DEMU の遅延時間（単位：µs）

挿入した遅延時間	0	1	10	100	1,000	10,000	100,000	1,000,000
最大値	56.7	56.9	62.1	154.4	1,047.8	10,042.8	100,042.1	1,000,051.2
最小値	9.4	10.0	18.1	107.5	1,007.3	10,007.1	100,008.0	1,000,015.8
平均値	11.3	12.3	20.5	109.6	1,010.0	10,010.0	100,010.2	1,000,018.2
標準偏差	2.0	2.2	2.2	2.3	3.4	3.3	2.8	2.8
再現誤差	—	0	0.8	1.7	1.3	1.3	1.1	6.9

表 4 10Gbps，パケット長 1500B での netem の遅延時間（単位：µs）

挿入した遅延時間	0	1	10	100	1,000	10,000	100,000	1,000,000
最大値	1,260.4	2,139.3	2,089.2	2,229.8	3,237.2	13,242.0	104,021.2	1,004,531.3
最小値	79.7	85.9	83.5	185.9	1,071.2	10,068.9	100,074.9	1,000,090.8
平均値	863.6	1,745.6	1,659.3	1,803.4	2,784.7	11,852.1	101,989.5	1,001,972.8
標準偏差	107.8	216.7	206.0	212.7	223.5	242.0	285.9	431.3
再現誤差	—	881.0	785.7	839.8	921.1	988.5	1,125.9	1,109.2

表 5 1Gbps，パケット長 1500B での netem の遅延時間（単位：µs）

挿入した遅延時間	0	1	10	100	1,000	10,000	100,000	1,000,000
最大値	132.1	147.7	152.3	205.1	1,128.6	10,131.6	100,131.5	1,000,146.9
最小値	8.4	11.6	20.0	109.4	1,010.3	10,009.3	100,010.0	1,000,017.1
平均値	42.1	42.1	58.4	148.6	1,048.5	10,045.8	100,048.1	1,000,053.8
標準偏差	18.9	17.2	21.1	21.3	21.3	20.8	21.0	20.7
再現誤差	—	1.0	6.3	6.5	6.4	3.7	6.0	11.7

表 6 1Gbps，パケット長 46B での netem の遅延時間（単位：µs）

挿入した遅延時間	0	1	10	100	1,000	10,000	100,000	1,000,000
最大値	5,484.9	13,267.4	11,878.7	12,737.4	13,691.8	23,894.7	127,789.9	1,030,244.2
最小値	76.3	83.9	80.0	185.1	1,080.3	10,085.6	100,096.5	1,000,116.0
平均値	3,036.2	7,295.7	6,585.1	7,071.1	7,938.5	17,661.1	108,479.5	1,008,156.2
標準偏差	1,383.3	3,323.6	2,995.7	3,176.4	3,161.2	3,492.0	3,881.7	3,882.2
再現誤差	—	4,258.5	3,538.9	3,934.9	3,902.3	4,624.9	5,443.3	5,120.0

97 %改善できた．遅延のエミュレーションの精度は，1 ms の遅延挿入時で，再現誤差は DEMU で 0.3 µs と高精度だったのに対し，netem は 921 µs とエミュレーションしたい 1 ms とほぼ同じであった．また，ショートパケットのような高パケット送信レートでも，DEMU はパケット長 1500B と同様に高精度にエミュレーションできた．DEMU 全体のオーバヘッドは，GtrcNET10-p3 の NIC1 と NIC2 を直結して評価したところ遅延時間が 0.98µs であったことから，およそ 10µs 程度であった．Linux では，文献 [3], [4] でも述べられているように，パケット処理のスループットが低いため，1Gbps においても遅延のジッタが大きく，netem のエミュレーション精度も劣っていた．

図 5 は，DEMU におけるパケット長 1500B での通信途中の 10,000 パケット分の遅延時間の推移である．このように，定期的に遅延時間が 16 から 23 µs に増加する傾向が見られた．NIC の送信処理において，複数のパケットを送信用リングバッファに貯めてから一度に送信する傾向が生じるためである．伝送帯域幅に近いトラフィックの場合は，送信処理に負荷がかかり NIC が保持するリングバッファで 6 から 20 µs 程度の遅延が入る．なお，1Gbps のト

© 2016 Information Processing Society of Japan

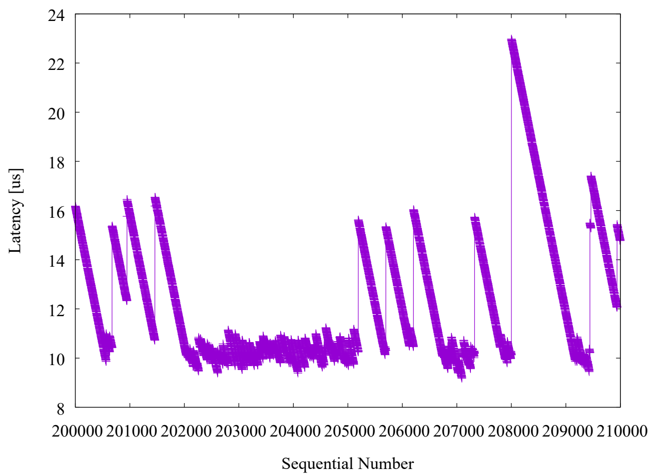

図 5 DEMU における遅延時間の推移

ラフィックのように，伝送帯域幅に余裕がある場合は，リングバッファでの待機処理が起こりにくいため，標準偏差が 0.2 µs になることを確認している．そのため，NIC の処理限界に近いトラフィックにおいて，レイテンシセンシティブなエミュレータを実現するには，送信処理を 1 パケット毎に処理するようにし，処理できなかったパケットに関しては破棄するようにすることで実現できる．

4. 関連研究

一般的に，ソフトウェアによるネットワークエミュレータは，Linux にデフォルトで搭載されている netem[1] が代表的である．しかし，netem は，本論文での評価結果からもわかるように，遅延時間のジッタが大きく，また 10Gbps のトラフィックではマイクロ秒オーダの制御が必要なレイテンシセンシティブなシステムには適用できない [5].

文献 [9] は，代表的なパケット処理フレームワークである netmap[10] を用いたネットワークエミュレータ TLEM について述べている．TLEM は，ショートパケットのトラフィックである 18Mpps や 40Gbps のトラフィックでもエミュレーション可能としている．しかし，文献 [9] の TLEM の評価では，遅延エミュレーションに関して詳細には記述されていない．文献 [8] には，netmap と DPDK とを比較解析しており，DPDK の方がスループットや遅延時間が優れていると述べている．そのため，今回開発したネットワークエミュレータの方が遅延のエミュレーションの精度では優れていると思われる．文献 [8] には，netmap や DPDK の他に，PF_RING ZC[11] も比較解析しており，処理性能面で DPDK に一定の優位性があることを報告している．提案システムと同様のネットワークエミュレータを，DPDK 以外のパケット処理フレームワークにおいても実装可能であると考える．

5. おわりに

本論文では，Intel DPDK を用いたネットワーク遅延エミュレータ DEMU について述べた．DEMU は，Intel DPDK を用いることで，Linux などオペレーティングシステムのネットワークスタックを経由しないため高速なパケット処理が可能である．DEMU では，受信スレッド，遅延をエミュレーションするワーカスレッド，送信スレッドの 3 つのスレッドに対して，それぞれ CPU コアを占有させコンテキストスイッチを抑制することで，ジッタの小さい伝送遅延をエミュレーションする．また，評価結果から DEMU は，パケット送信レートの高い 10GbE のショートパケットのトラフィックに対しても，ジッタが小さく，高精度な伝送遅延のエミュレーションが可能であることを確認した．

今後の予定は，伝送遅延以外に，帯域制限やパケットロスなど様々なネットワークのエミュレーションを実現する．

謝辞 本研究は，文部科学省イノベーションシステム整備事業の補助による「光ネットワーク超低エネルギー化技術拠点（VICTORIES 拠点）」での研究成果の一部を用いている．

参考文献

[1] Linux Foundation Wiki: `https://wiki.linuxfoundation.org/networking/netem` (2016).
[2] The dummynet project: `http://info.iet.unipi.it/~luigi/dummynet/` (2016).
[3] Emmerich, P., Raumer, D., Wohlfart, F. and Carle, G.: Assessing soft-and hardware bottlenecks in PC-based packet forwarding systems, *IARIA ICN 2015* (2015).
[4] Bolla, R. and Bruschi, R.: Linux software router: data plane optimization and performance evaluation, *Journal of Networks*, Vol. 2, No. 3, pp. 6–17 (2007).
[5] Larsen, S., Sarangam, P., Huggahalli, R. and Kulkarni, S.: Architectural Breakdown of End-to-End Latency in a TCP/IP Network, *International Journal of Parallel Programming*, Vol. 37, No. 6, pp. 556–571 (2009).
[6] DPDK: Data Plane Development Kit: `http://dpdk.org/` (2016).
[7] 児玉祐悦，工藤知宏，清水敏行：10GbE 対応ネットワークテストベッド GtrcNET-10 の構成と評価，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2005, No. 81, pp. 109–114 (2005).
[8] Gallenmüller, S., Emmerich, P., Wohlfart, F., Raumer, D. and Carle, G.: Comparison of Frameworks for High-Performance Packet IO, *Proc. of the Eleventh ACM/IEEE Symposium on ANCS '15*, pp. 29–38 (2015).
[9] Rizzo, L., Lettieri, G. and Maffione, V.: Very high speed link emulation with TLEM, *IEEE International Symposium on LANMAN '16*, pp. 1–6 (2016).
[10] Rizzo, L.: Netmap: A Novel Framework for Fast Packet I/O, *Proc. of USENIX ATC'12*, pp. 101–112 (2012).
[11] ntop: PF_RING ZC (Zero Copy): `http://www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/` (2016).

© 2016 Information Processing Society of Japan