Vol.2018-OS-144 No.5
2018/7/30

Rump kernelの通信性能の評価と改善手法の検討

胡 思已[1,a)]　坂本 龍一[1]　近藤 正章[1]　中村 宏[1]　新 善文[2]

概要：
Linux 系の OS は積極的に開発されており，デバイスドライバの実装スピードも早く，最新のデバイスを安定動作させることが可能であることから，様々な機器に搭載される OS として広く用いられている．一方で，NetBSD は安全で高い移植性を備えた OS であり，アプリケーションやプロトコルスタックの実装にも頑健性があるなど優れた点も多い反面，Linux に比べて最新のデイバスをサポートしていないなどの課題もある．そのため，最新のデバイスにおいて NetBSD がサポートする頑健なプロトコルスタックを用いることは容易ではない．そこで，NetBSD のカーネルをユーザレベルのプロセスとして実装し，NetBSD のプロトコルスタックやアプリケーションを Linux から利用するための Rump kernel が開発されている．これまで，Rump kernel 利用時の通信性能については十分に評価がされておらず，また通信性能の改善に関しても十分に検討されてこなかった．そのため，本稿ではまず，Rump kernel 利用時の通信性能を評価した．その結果，ネイティブな Linux の通信性能に比べて非常に低い性能しか得られないことがわかった．そこで，Rump kernel 利用時の通信性能の向上手法を検討し，実装を行った．初期実装版で評価を行ったところ，Rump kernel を利用したデータ送信では，95.9%程度の性能向上が得られることがわかった．

1. はじめに

近年，多くの開発者によって Linux カーネルの性能向上や新しい機能への対応が盛んに進められている．また，積極的に新しいデバイスへの対応が行われており，Linux は最新の多くのデバイスでも動作する．この結果，Linux カーネルは，世界中のウェブサーバーやスマートフォン等の組み込み機器など幅広い分野において利用されている．

一方，Linux のデメリットとして安全性の問題やカーネルエラーによってシステム全体が不安定になるといった課題がある．近年では度々深刻な貧弱性が報告されている．さらにソースコードが日々アップデートされるため，移植性，スケーラビリティなどの要求には不向である．故に安全性が厳しく要求されるドメイン，またはスケーラビリティが求められるシステムでは Linux 以外の信頼性の高い OS が選択されるケースも多くある．

Linux と同じ UNIX 系の OS である Berkeley Software Distribution（BSD）シリーズは，安全性，高い移植性などのメリットを持ち，様々な組み込み機器，特にネットワーク通信機器によく使われている．特に NetBSD は 386BSD から派生し，正式にリリースされた最初のオープンソース BSD ディストリビューションである．NetBSD プロジェクトは，コードの明確さ，慎重な設計，および多くのアーキテクチャにわたる移植性に重点が置かれている．主に簡潔なデザインによる高い安全性，高いスケーラビリティという特徴を備えている．セキュリティに関するバグ報告も他の OS と比較すると非常に少ない．そのため高い安全性・信頼性が求められる大規模なサーバシステム，デスクトップシステム，ハンドヘルドデバイスなど，多くのプラットフォームで NetBSD は使用されている．

しかし，NetBSD は Linux に比べて最新のデイバスをサポートしていないなどの課題がある．そのため，高い安全性と最新デバイスへの対応を両立させることは容易ではない．そこで，NetBSD のカーネルを Linux のユーザレベルのプロセスとして実装し，NetBSD のプロトコルスタックやアプリケーションを Linux 上で利用するためのソフトウェア環境として Rump kernel が開発されている [5], [6]．ネットワーク処理やカーネル処理は LibraryOS 化した NetBSD のコードを利用して実行される．さらに，tap デバイスを用いてホスト OS である Linux と接続することで，Linux がサポートする最新のネットワークインタフェースカードを利用することができる [2]．このような機能を提供することで Rump kernel は NetBSD による安全性の確保と Linux による様々なデバイスへの対応という要求を同時に満たすことが可能となる．

一方で，Rump kernel の通信性能については十分な評価

[1] 東京大学大学院情報理工学系研究科
[2] アラクサラネットワークス株式会社
[a)] hu@hal.ipc.i.u-tokyo.ac.jp

がなされておらず，また通信性能の改善に関しての検討もあまり行われていない．安全性のみならず，高性能な通信環境を提供することも重要である．そこで，本研究ではまず Rump kernel の通信性能を評価した．その結果，ネイティブな Linux の通信性能に比べて非常に低い性能しか得られないことがわかった．十分に大きなバッファサイズを用いて通信する場合においても，Linux と比較して，10.3%の送信性能と 24.7%の受信性能しか出ない結果となった．これらの結果を踏まえ，次に Rump kernel 利用時の通信性能の向上手法として，メモリ管理方式の実装の改善手法を検討した．初期実評価を行ったところ，Rump kernel を利用したデータ送信では従来と比較して，95.9%の通信性能向上が得られることがわかった．

2. Rump kernel の概要

Rump kernel プロジェクトは，例えば LibraryOS として知られるように，NetBSD をポータブルなソフトウェアスタックとして利用可能にし，様々なハードウェアや実行環境上で NetBSD のリソースを提供することを目指したシステムソフトウェアプロジェクトである [11]．このために，Anykernel と呼ばれるコンセプトを採用している．本章では，Rump kernel の概要について述べる．

2.1 Anykernel

Anykernel は，ソースコードを修正することなく，様々な実行環境でカーネル機能を利用することを目指して定義されたコンセプトである．Anykernel として利用可能な NetBSD は，アプリケーションライブラリやマイクロカーネル上のユーザプログラムとしてカーネルコードを実行することが可能である．また，モノリシックカーネルの一部としても動作する．Anykernel では，カーネルコード内のファイルシステムやネットワークスタックも LibraryOS として利用できる．Rump kernel は Anykernel のコンセプトを NetBSD カーネル向けに実装したものであり，NetBSD のカーネルを Unikernel としてベアメタルマシン上で直接実行したり，Linux のプロセスとして動作させることができる [3], [4]．以下に，Rump kernel が提供する 2 つの実行方式について述べる．

- ベアメタルマシン上で実行可能な Unikernel
  ユーザーアプリケーション，ライブラリ，NetBSD カーネルを 1 つのイメージとし，ベアメタルマシン上で実行する方式である．プログラムを Unikernel として動作させるため，非常に軽量に動作する．また，比較的新しいハードウェアもいくつかサポートされており，ブートローダである GRUB から直接実行可能である [8]．さらに Amazon AWS などのクラウド VPS 向けの起動イメージを生成する機能もあり，本 Unikernel として実装したアプリケーションをクラウド上でも簡単に利用可能である．
- クライアントサーバモデル
  NetBSD 資源を Linux が提供するプロセスとして実行するモデルである．ユーザーアプリを実行する Rump client と NetBSD のカーネル部分を担う Rump server から構成される．ユーザアプリケーションは Rump client として動作し，必要に応じて Rump server と通信を行う．これらのクライアントとサーバは標準的なプロセスとして実行できるように設計されており，様々な Linux 環境上で動作可能である．このため，Ubuntu や CentOS だけでなく，Cygwin などの環境でも容易に動作する．しかし，後に述べるように Rump client，Rump server 間でのデータコピーのオーバーヘッドが課題となる．

以降，本稿では特にクライアントサーバモデルに着目する．クライアントサーバモデルは既存の Linux がサポートする最新のネットワークインタフェースカードが利用可能であり，かつ高い安全性が得られるという利点があるためである．

2.2 クライアントサーバモデル

クライアントサーバモデルにおける Rump kernel の概要を図 1 の左側に示す．本節では，通信部分に特化して説明する．

Rump kernel のユーザーアプリケーションはホスト OS のプロセスである Rump client として動作する．メモリ管理やスレッド処理等の機能はユーザーアプリケーションの中で直接実行されるが，通信等のシステムコールは librumpclient によってハンドリングされ，NetBSD の機能が利用される．librumpclient はホスト OS が提供する IPC (Inter-Process Communication) を用いて Rump server に要求を伝える．Rump server もプロセスとして動作しており，HostOS が提供する IPC によって Rump client からの要求を受け取る．その後，ユーザプロセスで動作する Rump server 内の BSD のプロトコルスタックを用いて通信の処理を行う．最終的に，ホスト OS に接続された仮想 NIC を通してデータの送信が行われる．このようにすることで，NetBSD が提供する高信頼性な BSD Protocol Stack を利用することができ，また Linux がサポートする最新の物理 NIC を利用することが可能となる [12]．一方で，Rump client，ホスト OS，Rump server 間でデータのコピーが生じるため，このデータコピーのオーバーヘッドが問題となる可能性がある．

3. 初期通信性能評価

Rump kernel として動作する NetBSD カーネルは Rump client，ホスト OS，Rump server 間でのデータコピーが頻繁に発生するため，Linux カーネルと比べて通信のスルー

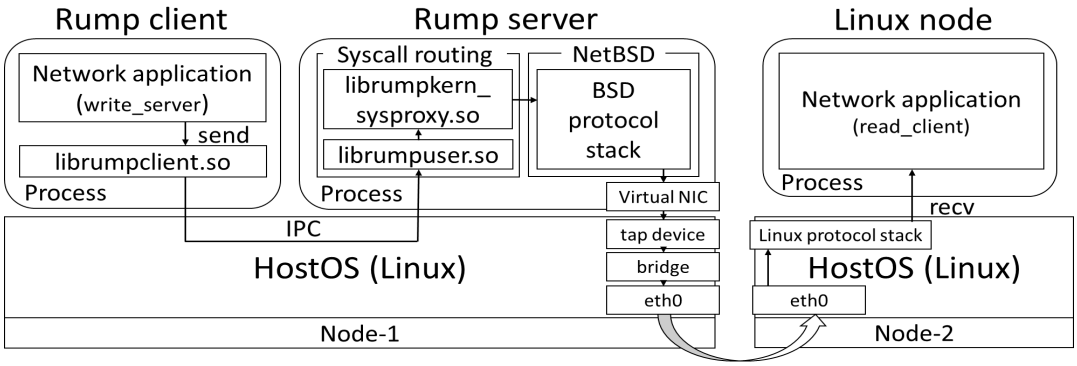

図 1　Rump kernel の概要と評価システム

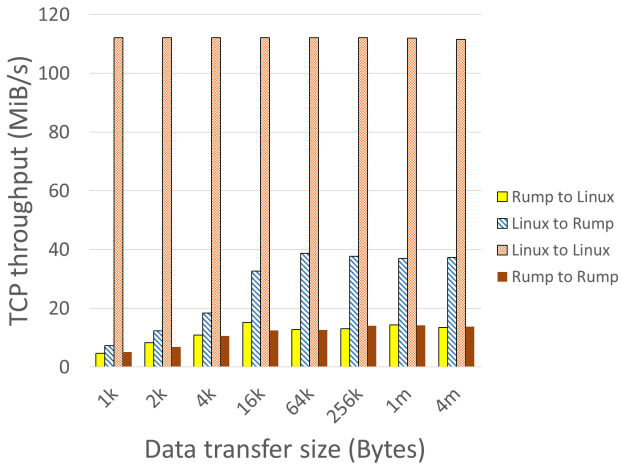

図 2　通信性能評価結果

プットが低下する可能性がある．この影響を定量的に評価するため，本章ではネットワーク通信性能に関してネイティブな Linux と通信のスループットを比較する．

3.1　評価内容

評価では 2 台の PC 間の通信速度について調査を行う．双方をネイティブな Linux とした場合，片方を Rump kernel とした場合，双方を Rump kernel とした場合について評価を行う．評価システムの全体像を図 1 に示す．なお，本図は送信側を Rump kernel とした場合を示している．

評価では，一方のノードから他方のノードに対してデータを送り続けるという評価アプリを自作した．Rump kernel 向けに netperf が移植されているが，不具合があり今回は使用していない．データ転送サイズを変えつつスループットの計測を行った．クライアントである read_client (右側，Linux node 中) はサーバーに対しデータ送信要求を送信する．要求を受けたネットワークアプリケーションである write_server (左側，Rump client 中) は read_client に対してデータを送り続ける．評価プログラムは C 言語で作成し，プログラムのコンパイル時にネイティブ Linux 向けのバイナリと Rump kernel 向けの 2 つのバイナリのを用意した．以降では，5 回測定した平均値を示す．

表 1　評価環境

CPU	Intel(R) Celeron(R) CPU J3160 @ 1.60GHz
RAM	8GB DDR3
NIC	Realtek RTL8111
	PCI Express Gigabit Ethernet Controller
ホスト OS	Ubuntu 16.04 LTS

3.2　評価環境

本評価では表 1 に示す PC を 2 台用いる．また，汎用のギガビットスイッチを用いてこの 2 台を接続する．多くのネットワーク機器では，消費電力に制約があることが多いため，組み込み向け CPU が利用されることが多い．そのため，本評価では省電力な Celeron プロセッサを用いて評価している．

3.3　評価結果

評価結果を図 2 に示す．通信性能を定量的に述べるため，以下では全バッファサイズのスループットの平均値を用いて比較する．ネイティブ Linux (図中は Linux と表記) と比べると，Rump kernel を用いた場合は送信スループットが 10.3%となり，受信スループットも 24.7%程度しか得られていない．双方をネイティブ Linux とした場合は，バッファサイズが小さい場合であっても 112MiB/s 程度の速度が得られており，ワイヤーレートに近いスループットが達成されていることがわかる．一方で，片方を Rump kernel にした場合，バッファサイズが 1K バイトの際には 7MiB/s 程度の性能しかでていない．バッファサイズを増加させた場合でも，多少の速度向上はあるものの，双方がネイティブ Linux の場合と比較すると相当に低速である．さらに，双方を Rump kernel にした場合，バッファサイズが 1K バイトの場合で 5MiB/s，バッファサイズを 4MiB/s としても 14MiB/s 程度の性能しか得られなかった．

4.　速度低下の原因の解析

本章では，Rump kernel 内のる NetBSD プロトコルスタックに着目し，通信性能低下の原因について述べる．まず，カーネルメモリの扱い方について説明し，送信時の内

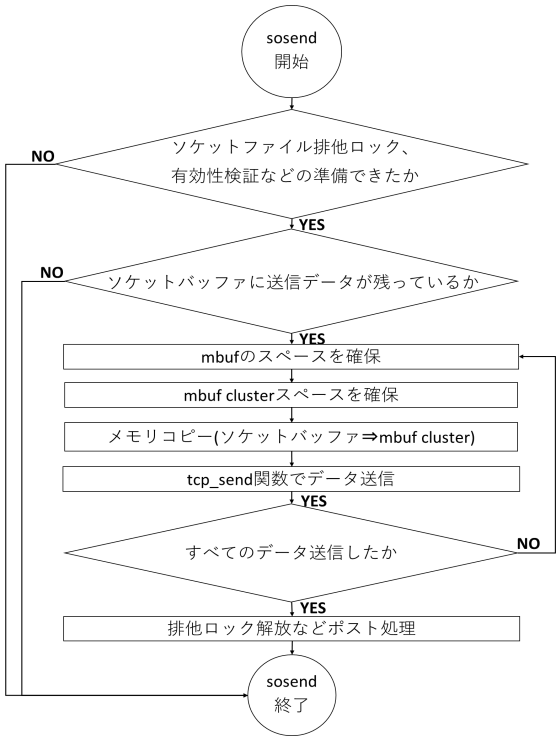

図 3　sosend 関数の処理の流れ

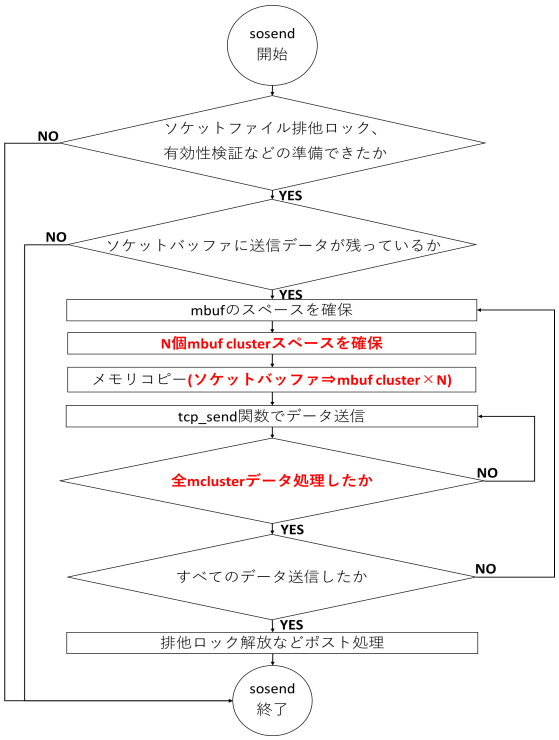

図 4　提案手法の sosend 関数の処理の流れ

部動作を説明する．そして，性能低下の原因を述べる．

4.1　カーネルメモリの取り扱い

NetBSD では起動時にカーネルコードで使用する多くのメモリの確保を行い，メモリプールとして管理する．これにより，実行時のメモリ確保のオーバヘッドを削減し，システムの高速化を目指している．これらメモリプールの中でも，特に mbuf と mbuf cluster がプロトコルスタックに対して重要な役割を担っている [1], [10].

mbuf mbuf は 512 バイトの固定長バッファをポインタで結んだ連結リスト形式のバッファである．ポインタで結んだ 1 つのリスト列を「mbuf チェーン」と呼び，複数のチェーンをまとめて，1 つの「キューレコード」を構成する．プロトコルスタックの中では，mbuf がネットワーク送受信データを格納するバッファの役割を持っている [7], [9].

TCP/IP において，ネットワークからデータパケットを受信した際に，下位のレイヤから順に Ethernet → IP → TCP レイヤとデータの処理が遷移する．下の層から上位の層へ到達するまでに，それぞれヘッダの除去，断片化されたパケットの再構築などの作業が行われる．

送信の場合は，下位層にデータを受け渡す際に，ヘッダの付加やデータの分割などが必要になる．そのため，何度もバッファを確保してデータをコピーしたり，不要になったバッファを解放したりする必要が生じる．このオーバーヘッドを抑止するため，NetBSD ではメモリプールを使い mbuf のメモリ空間を管理している．

mbuf cluster ネットワーク通信では大きなサイズのデータを送受信する場合もあるため，mbuf 内部でデータを保存できない場合もある．そのため，上記で述べた外部メモリ領域を参照するタイプの mbuf がよく使用される．mbuf が参照する外部メモリ領域は mbuf cluster として定義される．mbuf cluster のメモリスペースもメモリプールから取得することであり，1 つのバッファのメモリサイズがデフォルトで 2048 バイト，その最大値はページサイズと同じ値である．

4.2　送信処理

Rump kernel では NetBSD プロトコルスタックのソケットレイヤ関数を用いて，データ処理が行われる．送信の場合には sosend が用いられる．sosend 関数の処理の流れを図 3 に示す．

ここで，456 バイト以上のデータを送信する場合には，mbuf cluster が使用され，mbuf cluster 上に保存されたデータがドライバに渡され NIC を通して送信される．

ソケットバッファから mbuf cluster までのデータコピーは，ユーザ空間からカーネル空間へのデータコピーである．ネイティブな OS ではトラップ指令を発行し，特権モードで本処理が実行される．Rump kernel の場合，Rump server プロセス (NetBSD) 側からリクエストを Rump client に渡し，コピーすべきデータを domain socket に通じて Rump client から転送する．ここで，このデータ転送のオーバーヘッドが性能低下の原因と考えられる．

5. 通信速度の改善手法

前章で述べた通り，ソケットバッファから mbuf データ領域に対して送信データのメモリコピーが必要となり，Rump client と NetBSD を搭載する Rump server の間にソケット通信としてデータ転送が行われている．データ転送量を変更することは難しいが，本稿では転送する回数を減らすことでオーバーヘッドの削減を狙う．

5.1 mbuf cluster サイズの拡張

利用した NetBSD のバージョンでは，mbuf cluster 領域のサイズはデフォルトで 2048 バイトに設定されている．1 回でコピーするデータの量を増大させることで，合計のコピー回数を削減することができ，データ転送により生じるオーバーヘッドを抑止することができると考えられる．なお，mbuf cluster のサイズは，ハードウェアアーキテクチャ毎に該当するヘッダファイルで定義されている．

5.2 コピー回数削減手法

4.1 節で述べたように，mbuf cluster の最大サイズはシステムのページサイズを超えることができない．HugePages 機能を有していない NetBSD では，ページサイズは 4096 バイト固定であるため，mbuf cluster の最大値も 4096 バイトとなる．そこで，より 柔軟にバッファサイズを変更できるようにするために，拡張 mbuf cluster を提案する．本手法は複数個の mbuf cluster をまとめて確保し，1 回のコピーリクエストでコピーできるデータ量を増加させるものである．本手法による sosend 関数の処理の流れを図 4 に示す．

起動時に mbuf と mbuf cluster 領域は必要なサイズ分を確保する必要がある．そのため，本手法では，複数の mbuf cluster メモリプールを一括で確保する．この際，1 回で送信できるソケットバッファのデータ容量に応じて，mbuf cluster を複数個確保することになる．mbuf cluster へのメモリコピーの際には，カーネル側がソケットバッファの中の全データを要求し，複数の mbuf cluster 領域にまとめてコピーする．そのため，提案手法を用いることで，mbuf cluster へデータコピー回数を最小回数まで削減できる．

最終的に，TCP レイヤの tcp_send 関数を用いてデータを送信する場合，上記の複数 mbuf cluster の領域を順次使用してデータを送り出す．

6. 通信性能向上効果の評価

本章では，提案手法を用いた際の Rump kernel の通信スループットを評価する．3 章で行った評価方法と同様に通信性能の評価を行うが，加えて mbuf cluster サイズなどのパラメータを変更しつつ評価を行う．

6.1 パラメータ

本提案手法には以下の 2 つのパラメータが存在する．

(1) MCLBYTES: mbuf cluster のサイズ．デフォルトは 2K バイトであり，最大はページサイズと等しい 4K バイトである．

(2) MCLNUM: 拡張 mbuf cluster の個数．mbuf cluster をまとめて確保する際のバッファの個数であり，デフォルトは 1 である．これを 4，8，16 と変化させる．

評価では，これら二つのパラメータを変えた場合について結果を示す．

6.2 mbuf サイズを変えた場合のスループット

Rump kenerl の送信スループットと受信スループットの測定結果を図 5 に示す．左のグラフが Rump kenerl からネイティブ Linux へ通信した際の送信スループットを，右のグラフがネイティブ Linux から Rump kenerl へ通信した際の受信スループットを示している．ここでは，mbuf サイズがディフォルトの 2048 バイト (MCLBYTES=2018) の場合と，4096 バイト (MCLBYTES=4096) に変更した場合の結果を示している．送信スループットについてデフォルトの場合と比較して 40.3%ほど向上したが，受信スループットは本来の 66.1% まで下がってしまう結果となった．送信スループットの向上は，mbuf サイズを大きくしたことで mbuf へのコピー回数が減り，そのオーバーヘッドが削減されたためと考えられる．

受信スループットについては TCP レイヤから受信するデータの量は常に MTU の最大値である 1446 バイトであり，2048 バイト以上の mbuf 領域を確保しても，1446 バイトしか使用されないことから，mbuf cluster のサイズ (MCLBYTES) を 4096 にする効果はない．一方で，この際に受信スループットが低下してしまう原因であるが，カーネルメモリプールの断片化によりメモリの使用効率が悪くなったことなどが考えられる．ただし，詳細な原因は不明であり，今後詳細に調査を行う予定である．

6.3 拡張 mbuf cluster 数を変えた場合のスループット

6.3.1 mbuf cluster サイズが 4096 バイトの場合

図 6 に，mbuf サイズを 4096 バイト (MCLBYTES=4096) に固定し，mbuf cluster 数を MCLNUM=1, 4, 8, 16 と変更した場合の送信スループット (左図) と受信スループット (右図) の測定結果を示す．なお，参考のために mbuf サイズを 2048，かつ MCLNUM=1 の場合も点線で示している．評価結果より，送信スループットについては，mbuf 数 (MCLNUM) を増やすことでスループットの向上効果があることがわかる．MCLNUM=16 の場合では，拡張前の Rump kernel (MCLBYTES=2048, MCLNUM=1) と比較して，スループットが最大 129.2%ほど向上した．一方で，受信スループットは 6.2 の結果と同様に本来の 66.1% まで

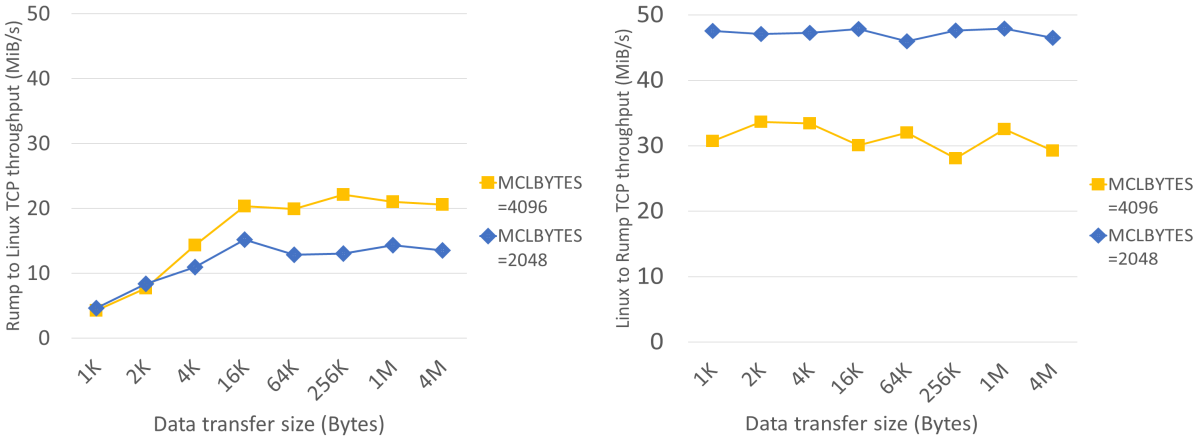

図 5 MCLNUM=1 の場合の通信スループット (左: 送信，右: 受信)

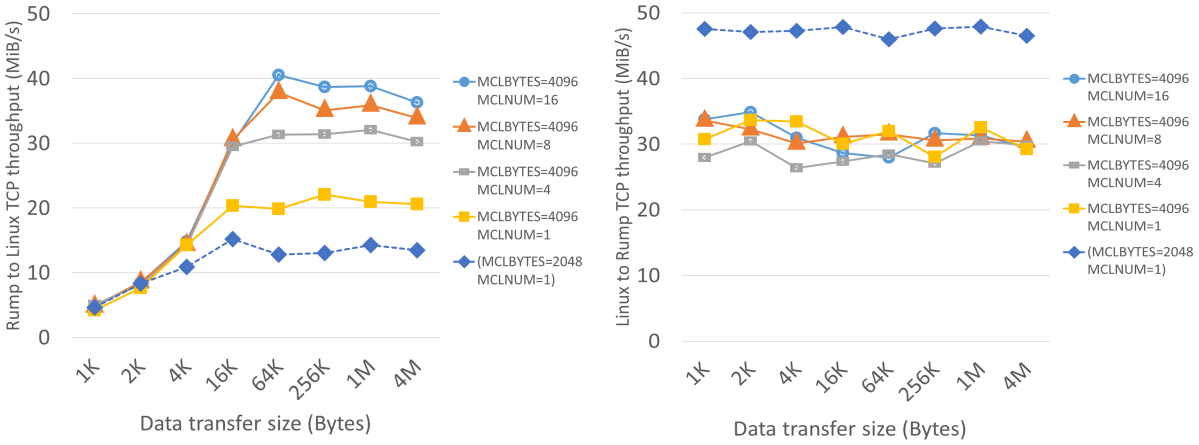

図 6 MCLBYTES=4096 の場合の通信スループット (左: 送信，右: 受信)

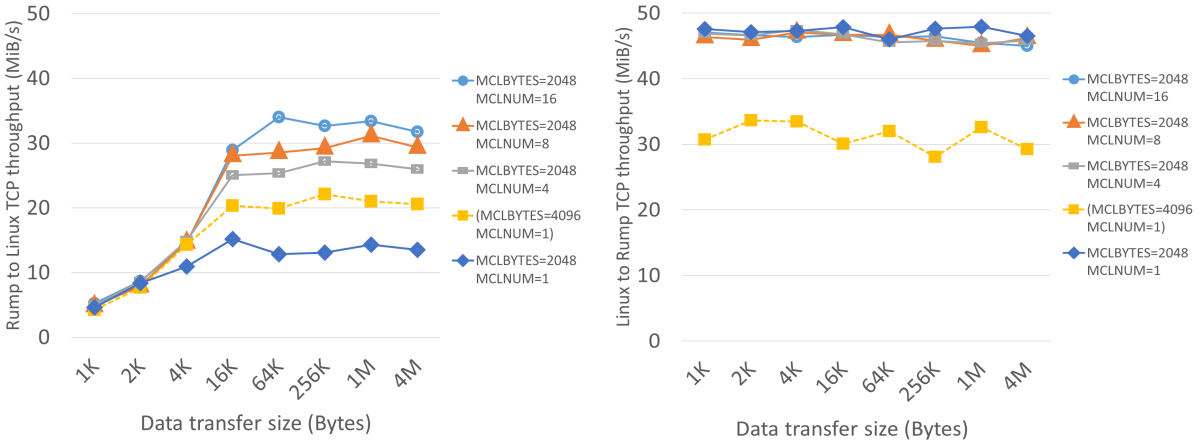

図 7 MCLBYTES=2048 の場合の通信スループット (左: 送信，右: 受信)

低下してしまうことがわかった．

6.3.2 2048 バイト mbuf cluster の場合

図 7 に，mbuf サイズを 2048 バイト (MCLBYTES=2048) に固定し，mbuf cluster 数を MCLNUM=1, 4, 8, 16 と変更した場合の送信スループット (左図) と受信スループット (右図) の測定結果を示す．なお，参考のために mbuf サイズを 4096，かつ MCLNUM=1 の場合も点線で示している．評価結果より，MCLBYTES=4096 の場合と同様に，MCLNUM=16 の時にデフォルトに対して最大 95.9%送信スループットが向上することがわかった．一方で，受信スループットは低下せず，拡張前の受信スループットと同じレベルを達成できれいる．

7. まとめと今後の課題

本稿では，NetBSD のカーネルをユーザレベルのプロセスとして実装し，NetBSD のプロトコルスタックやアプリケーションを Linux から利用するための Rump kernel について，通信性能の評価を行い，その改善手法に関して検討を行った．初期実装版で評価を行ったところ，Rump kernel を利用したデータ送信では 95.9%程度の性能向上が得られることがわかった．

今後の課題としては，受信性能の向上手法を検討すること，また遅延と NetBSD の長所でもある安定性に対しても評価を行うことなどがあげられる．

謝辞

本研究は，新エネルギー・産業技術総合開発機構からの委託研究「高効率・高速処理を可能とする AI チップ・次世代コンピューティングの技術開発（研究開発項目③，高度な IoT 社会を実現する横断的技術開発）『次世代産業用ネットワークを守る IoT セキュリティ基盤技術の研究開発』」の一部として行った．

参考文献

[1] Charles D. Cranor, Gurudatta M. Parulkar, “The UVM virtual memory system”, In Proceedings of the 1999 USENIX Annual Technical 168 BIBLIOGRAPHY 169 Conference (USENIX-99), pp. 117–130, Berkeley, CA, 1999. USENIX Association.
[2] Antti Kantee, “Environmental Independence: BSD Kernel TCP/IP in Userspace”, In Proceedings of AsiaBSDCon 2009, pp. 71–80, 2009.
[3] Arnaud Ysmal, Antti Kantee, “Fs-utils: File Systems Access Tools for Userland”, In Proceedings of the EuroBSDCon 2009, 2009.
[4] Antti Kantee: “Kernel Development in Userspace - The Rump Approach”, BSDCan 2009, 2009.
[5] Justin Cormack, “The rump kernel: A tool for driver development and a toolkit for applications”, AsiaBSDCon 2015, 2015.
[6] Antti Kantee, “The Design and Implementation of the Anykernel and Rump Kernels”, Aalto university, 2016.
[7] Xusheng Zhan, Yungang Bao, Christian Bienia, Kai Li, “PARSEC3.0: A Multicore Benchmark Suite with Network Stacks and SPLASH-2X”, ACM SIGARCH Computer Architecture News archive, Volume 44 Issue 5, December 2016, pp. 1-16.
[8] Kevin Elphinstone, Amirreza Zarrabi, Kent Mcleod, Gernot Heiser, “A Performance Evaluation of Rump Kernels as a Multi-server OS Building Block on seL4”, APSys ’17 Proceedings of the 8th Asia-Pacific Workshop on Systems, Article No. 11, 2017.
[9] Steven H. Rodrigues, Thomas E. Anderson , David E. Culler, “High-performance local area communication with fast sockets”, ATEC ’97 Proceedings of the annual conference on USENIX Annual Technical Conference, pp. 20-20, 1997.
[10] SungWon Chung, “The Design of the NetBSD I/O Subsystems”, Grin Publishing, 2016.
[11] NetBSD Wiki, “Rump kernel”, http://wiki.netbsd.org/rumpkernel/
[12] FOSDEM, “The Anykernel and Rump Kernels”, https://archive.fosdem.org/2013/interviews/2013-antii-kantee/