特 別 研 究 報 告 書

題 目

Mintオペレーティングシステムにおける仮想ネットワークインタフェースによるOSノード間通信の実現

指導教員

報 告 者

小倉 伊織

岡山大学工学部 情報系学科

平成30年2月9日 提出

要約

計算機の性能の向上に伴い，1 台の計算機上で複数の Linux を独立に走行させる方式として Mint が研究開発されている．Mint では，各 Linux に計算機資源を割り当て，直接占有制御する．Mint において，OS 間で通信を行う場合，NIC ハードウェアを用いて通信する手法と共有メモリを介して通信する手法の 2 つがある．しかし，これらの手法には問題がある．前者では，通信を行う OS それぞれに NIC ハードウェアを占有させる必要がある．Mint 上で同時に走行可能な OS の数は，CPU のコア数の増加に伴い増加しているため，用意する NIC ハードウェアの数も増やす必要がある．しかし，1 台の計算機に搭載可能な NIC ハードウェアの数には限界があるため，すべての OS に NIC ハードウェアを占有させることができない．このため，通信を行う OS ノードの数に限界がある．後者では，Mint 独自の通信インタフェースとプロトコルが必要であるため，通信を行うアプリケーションは，Mint の通信方式に対応していなければならない．このため，Linux 既存のアプリケーションは，この通信方式に合わせた改変が必要である．

そこで，本研究では，Mint における OS 間の通信を Linux 互換の形で実現する．具体的には，共有メモリを介した通信を Ethernet 互換で行う仮想ネットワークインタフェースとするデバイスドライバとして実現する．これにより，Linux 既存のアプリケーションを改変する必要がなくなる．評価の結果，本手法では，Linux 既存のアプリケーションを改変することなく OS 間で通信可能であり，実効スループットが 802$Mbps$ であることを示した．また，提案手法の実装のために追加したコード量が 474 行であることを示した．これは，Mint カーネル全体のコード量の約 0.004% であった．したがって，提案手法の実装は容易である．

目次

図 目 次

表 目 次

第 1 章

はじめに

計算機の高性能化に伴い，計算機資源をすべて使用することは少なくなってきた．そこで，計算機資源を有効活用するため，計算機の仮想化によって複数 OS を 1 台の計算機上で動作させる技術として，VMware[1] や Xen[2] といった仮想計算機方式が研究されている．しかし，これらの方式は，仮想化によるオーバヘッドのため，実計算機よりも性能が低下する．また，OS 間で処理負荷の影響が存在する．そこで，性能の低下を抑えるため，1 台の計算機上で複数の Linux を独立に走行させる方式として Mint （Multiple Independent operating system with New Technology）[3] オペレーティングシステムが研究開発されている．Mint では，各 Linux が計算機資源を割り当て，直接占有制御する．Mint において，OS 間で通信を行う場合，Network Interface Card（以降，NIC）ハードウェアを用いて通信する手法と共有メモリを介して通信する手法の 2 つがある．しかし，これらの手法には問題がある．前者では，通信を行う OS それぞれに NIC ハードウェアを占有させる必要がある．Mint 上で同時に走行可能な OS の数は，CPU のコア数の増加に伴い増加しているため，用意する NIC ハードウェアの数も増やす必要がある．しかし，1 台の計算機に搭載可能な NIC ハードウェアの数には限界があるため，すべての OS に NIC ハードウェアを占有させることができない．このため，OS 間で通信を行う OS ノードの数に限界がある．後者では，Mint 独自の通信インタフェースとプロトコルが必要であるため，通信を行うアプリケーションは，Mint の通信方式に対応していなければならない．このため，Linux 既存のアプリケーションは，この通信方式に合わせた改変が必要である．

そこで，本研究では，Mint における OS 間通信を Linux 互換の形で実現する．具体的には，共有メモリを介した通信を Ethernet 互換で行う仮想ネットワークインタフェースとするデバイスドライバとして実現する．これにより，Linux 既存のアプリケーションを改変する必要がなくなる．

本稿では，まず，Mint オペレーティングシステムにおける OS 間通信手法とその問題について述べる．次に，仮想ネットワークインタフェースによる OS 間通信手法の設計について述べる．その後，設計を実現する上での課題と対処について述べる．そして，仮想ネットワークインタフェースの実装について述べる．最後に，仮想ネットワークインタフェースを用いた OS 間通信の評価結果について述べる．

第 2 章

Mint における既存の OS ノード間通信とその問題点

2.1 Mint オペレーティングシステム

2.1.1 設計方針

Mint は，マルチコアプロセッサ上で複数の Linux を独立に走行させる方式である．Mint は以下の設計方針に基づき実現されている．

(1) すべての OS は互いに平等であり，依存関係が存在しない．

(2) すべての OS は相互に処理負荷の影響を与えない．

2.1.2 構成

図 2.1 に Mint の構成例を示し，CPU，メモリ，共有メモリ，および入出力機器について，各 OS による管理方法を以下で説明する．Mint では，最初に 1 つの OS を起動し，後から別の OS を起動する．最初に起動する OS を OS ノード 0 とし，後から起動する OS を起動順に OS ノード 1，OS ノード 2 とする．

(1) CPU
コア単位で分割し，各 OS ノードはコアを 1 つ以上占有する．

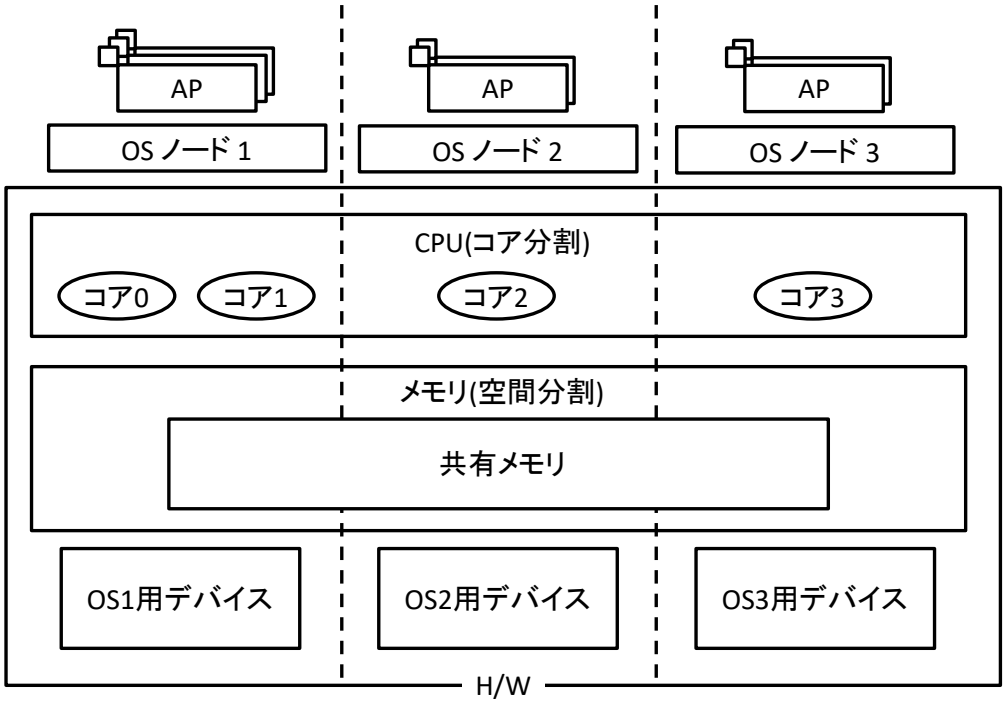

図 2.1 Mint の構成例

(2) メモリ
各 OS ノードのメモリマップを改変し，各 OS ノードに異なるメモリ領域を使用可能なメモリ領域として認識させる．

(3) 共有メモリ
各 OS ノードのメモリマップを改変し，各 OS ノードに共通のメモリ領域を使用可能なメモリ領域として認識させる．

(4) 入出力機器
デバイス単位で分割し，各 OS ノードは指定された入出力のみを占有する．

2.2 既存の OS ノード間通信

Mint における既存の OS ノード間通信は，2 つの手法で実現されている．この 2 つの手法について，以下で説明する．

(手法 1) NIC ハードウェアを用いた通信 (図 2.2)
OS ノード間通信を NIC ハードウェアを用いてネットワーク経由で行う手法である．

Mint では，計算機資源を各 OS ノードが分割/占有している．このため，この手法では，通信を行う OS ノードの数だけ NIC ハードウェアを用意し，それぞれの OS ノードで占有する．これにより，各 OS ノードは NIC ハードウェアと NIC ドライバを用いてネットワーク経由で OS ノード間通信が可能となる．この手法における OS ノード間通信の処理流れを図 2.2 に示し，以下で説明する．

(1) 送信側のアプリケーションは，write などのシステムコールを呼び出し，カーネル空間に送信するデータを渡す．

(2) 送信側のプロトコルスタックは，データをイーサネットパケットに変換する．

(3) 送信側の NIC ドライバは，イーサネットパケットを NIC ハードウェアのメモリ領域に書き込む．

(4) 送信側の NIC ハードウェアは，イーサネットパケットをネットワークを経由で受信側の NIC をハードウェアに送信する．

(5) 受信側の NIC ドライバは，NIC ハードウェアからイーサネットパケットを読み出す．

(6) 受信側のプロトコルスタックは，イーサネットパケットをデータに変換する．

(7) 受信側のアプリケーションは，read などのシステムコールを呼び出し，カーネル空間からデータを受け取る．

(手法 2) 共有メモリを介した通信 (図 2.3)

OS ノード間通信を Mint がもつ共有メモリを介して行う手法である．具体的には，Mint 独自の通信インタフェースとプロトコルを用いることで，Mint がもつ共有メモリを介して行う手法である．Mint がもつ共有メモリは，すべての OS ノードが読み書き可能である．このため，パケットを共有メモリを介してやり取りすることで OS ノード間通信が可能となる．この手法における OS ノード間通信の処理流れを図 2.3 に示し，以下で説明する．

(1) 送信側のアプリケーションは，Mint のプロトコルに対応したシステムコールを呼び出し，カーネル空間に送信するデータを渡す．

(2) 送信側の OS ノードは，Mint のプロトコルに則って，データを Mint の通信インタフェースに対応したパケットに変換する．

(3) 送信側の Mint の通信インタフェースは，パケットを共有メモリに書き込む．

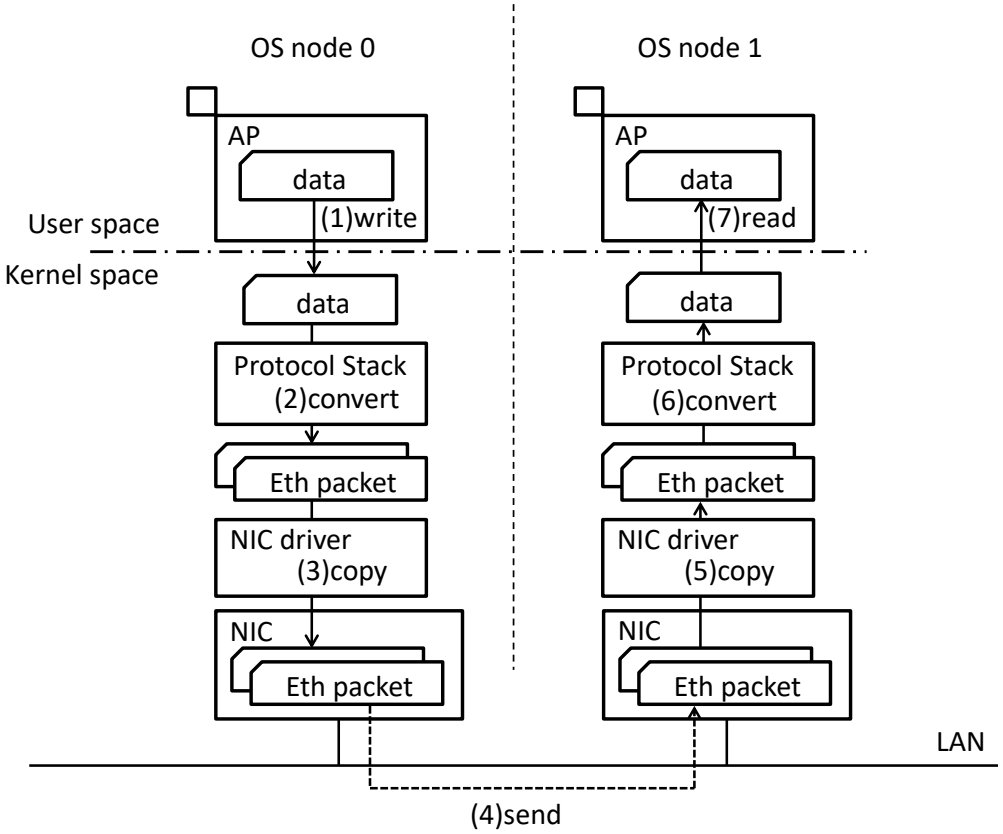

図 2.2 NIC ハードウェアを用いた通信

(4) 受信側の Mint の通信インタフェースは，共有メモリからパケットを読み出す.

(5) 受信側の OS ノードは，Mint のプロトコルに則って，パケットをデータに変換する.

(6) 受信側のアプリケーションは，Mint のプロトコルに対応したシステムコールを呼び出し，カーネル空間からデータを受け取る.

2.3 問題点

Mint における既存の OS ノード間通信手法が抱える問題について，以下で説明する.

(1) （手法 1）の問題点

(問題点 1-1) 通信を行う OS ノードの数に限界がある

（手法 1）では，通信を行う各 OS ノードに NIC ハードウェアを占有させる．このため，通信を行う OS ノードの数だけ NIC ハードウェアを用意しなければならな

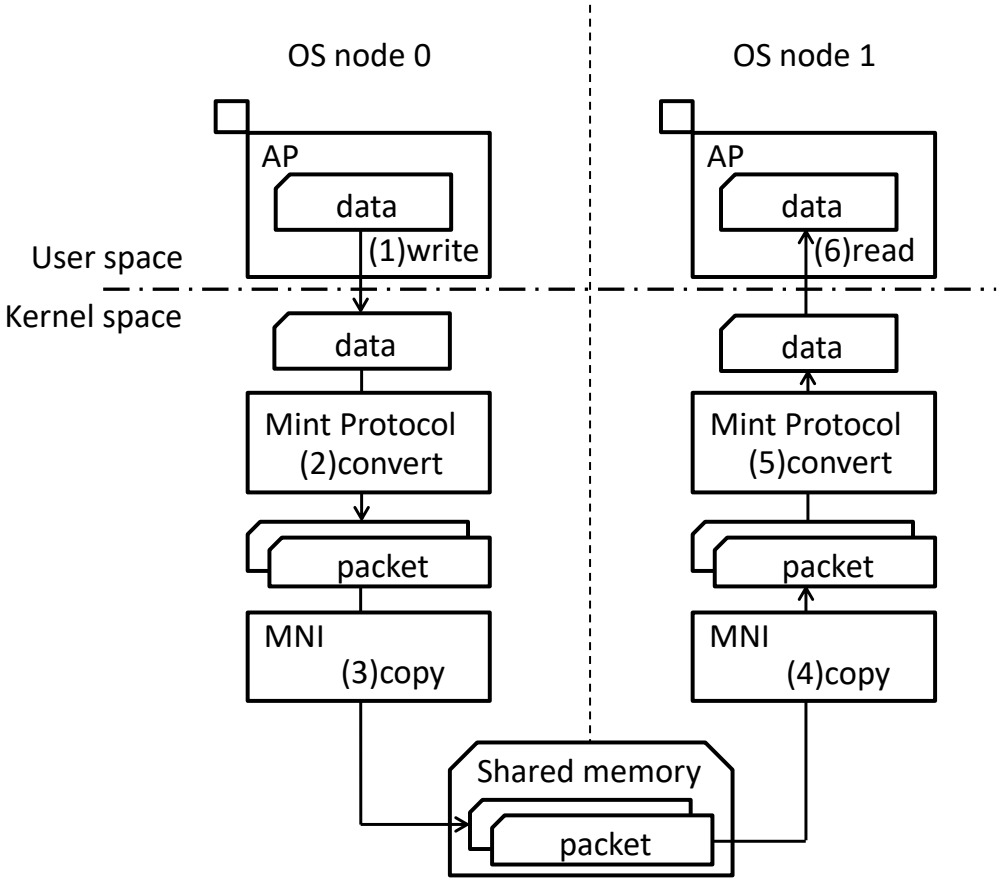

図 2.3 共有メモリを介した通信

い．Mint 上で同時に動作可能な OS ノードの数は，CPU のコア数の増加に伴い増加しているため，用意する NIC ハードウェアの数も増やす必要がある．しかし，1 台の計算機に搭載可能な NIC ハードウェアの数には限界があるため，すべての OS ノードに NIC を占有させることができない．このため，通信を行う OS がノードの数に限界がある．これは，すべての OS は互いに平等であるという Mint の設計方針に反するため，問題である．

(問題点 1-2) 異なる計算機上での通信と同程度のスループット

OS ノード間通信では，通信を行う OS ノードは同一計算機上で動作している．このため，OS ノード間通信では，通信のスループットを同一計算機上での通信と同程度にできる考えられる．しかし，(手法 1) では，OS ノード間通信を NIC ハードウェアを用いてネットワーク経由で行うため，通信のスループットは，異なる計算機上での通信と同程度となる．

(2) （手法 2）の問題点

(問題点 2-1) アプリケーションの改変が必要

（手法 2）では，通信を行うために Mint 独自の通信インタフェースとプロトコルを用いる．このため，通信に用いるアプリケーションはこの通信手法に対応していなければならない．しかし，Linux 既存のアプリケーション（以降，既存 AP）はこの通信手法に対応していない．このため，Mint 上で既存 AP を用いて OS ノード間通信を行う場合，既存 AP を改変しなければならない．通信を行う既存 AP の数は膨大であり，これらの AP すべてに改変を加えることは手間が大きい．

第 3 章

仮想ネットワークインタフェースによるOSノード間通信

3.1 設計方針

Mint における新たな OS ノード間通信手法（以降，提案手法）の設計方針について以下に示す.

(方針 1) 仮想ネットワークインタフェースでハードウェアの機能を代替

NIC ハードウェアを用いた通信では，通信を行う OS ノードの数に限界がある．そこで，OS ノード間通信を NIC ハードウェアを使用しない手法で実現する．このため，ソフトウェアによる仮想的なネットワークインタフェース（以降，仮想ネットワークインタフェース）を実現し，NIC ハードウェアの機能を代替する．NIC ハードウェアの機能の内，実割込みの発生は通信に不可欠である．そこで，仮想ネットワークインタフェースは IPI を用いて実割込みを代替する.

(方針 2) Mint がもつ共有メモリを介してパケットを送受信

NIC ハードウェアを用いた通信では，異なる計算機上でのプロセス間通信と同等のスループットを要する．そこで，パケットを計算機上の資源を介して受渡しすることで OS ノード間通信を行う．Mint において，すべての OS ノードが参照/更新可能な資源は共有メモリのみである．このため，パケットを共有メモリを介して受渡しする.

(方針 3) アプリケーションの改変が不要

共有メモリを介した通信では，既存 AP の改変が必要であり，改変にかかる手間が大

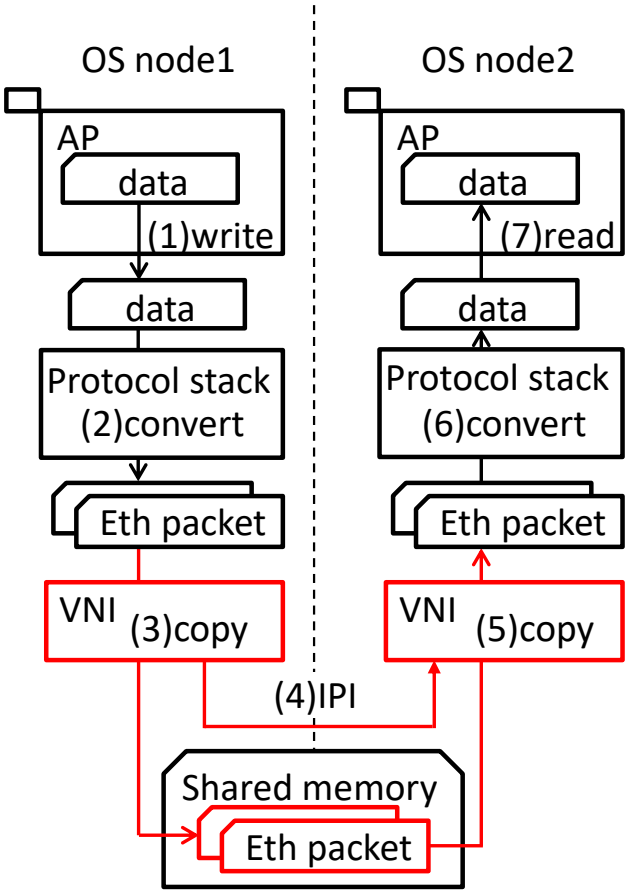

図　3.1　VNI を用いた OS ノード間通信

きい．このため，提案手法では既存 AP の改変なしに OS ノード間で通信可能とする．そこで，提案手法をユーザ空間とアプリケーションにカーネルのサブネットワークシステムにアクセスする手段を提供している上位層プロトコルに改変を加えず実現する．

3.2　設計

3.1 節から，提案手法を共有メモリを介して Ethernet 互換で OS ノード間通信を行う仮想ネットワークインタフェース（以降，VNI）として実現する．

VNI による OS ノード間通信について図 3.1 に示し，以下で説明する．

(1) 送信側の既存 AP は，write などのシステムコールを呼び出すことで送信するデータをカーネル空間に渡す．

(2) 送信側のプロトコルスタックは，データをイーサネットパケットに変換する．

(3) 送信側の VNI は，イーサネットパケットを共有メモリに書き込む.

(4) 送信側の VNI は，受信側の VNI に IPI を送信する.

(5) 受信側の VNI は，IPI をを受け取ると共有メモリからイーサネットパケットを読み出す.

(6) 受信側のプロトコルスタックは，イーサネットパケットをデータに変換する.

(7) 受信側の既存 AP は，read などのシステムコールを呼び出すことでカーネル空間からデータを受け取る.

3.3 課題

3.2 節の設計を実現する上での課題を以下に示す.

(課題 1) 送受信バッファの構成の検討

提案手法では，共有メモリを介して OS ノード間通信を行う．このため，共有メモリに対してパケットを書き込み/読み出しするための領域（送受信バッファ）を必要とする．送受信バッファの構成には，パケットを送信先別に分類して異なる送受信バッファで管理する構成と異なる送信先のパケットを同一の送受信バッファで管理する構成の 2 つが考えられるが，どちらが提案手法に適しているか分からない．このため，提案手法に適した送受信バッファの構成を検討する必要がある.

(課題 2) 排他制御すべき操作の検討

提案手法では，共有メモリを介して OS ノード間通信を行う．このため，複数の OS ノードが共有資源を同時に参照/更新することによる競合が発生すると考えられる．そこで，共有資源を参照/更新する操作にスピンロックを用いて排他制御する．これにより，共有資源の操作にはロックの獲得が必要となる．ロックを獲得できない場合，獲得できるまで単純なループを繰り返し，定期的にロックをチェックしながら待たせることで，競合の発生を防ぐことが可能である．しかし，ロックを保持する時間が長い場合，ロックを保持した状態でスケジューラによって割り込まれる可能性がある．このため，ロックを獲得するまでの待ち時間が長大化すると考えられる．このため，ロックを保持する時間が最小となるようにどの操作に排他制御を施すか検討する必要がある.

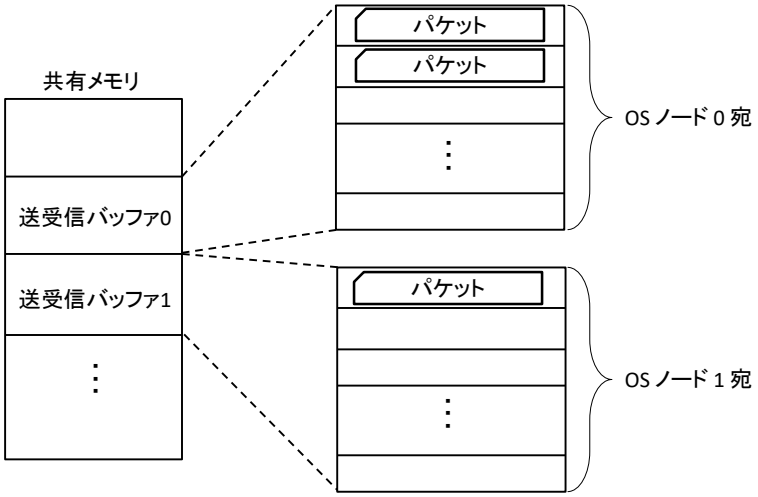

図 3.2 パケットを送信先毎に別々の送受信バッファで管理する構成

3.4 対処

3.4.1 送受信バッファの構成の検討

送受信バッファの構成には，パケットを送信先別に分類して異なる送受信バッファで管理する構成と異なる送信先のパケットを同一の送受信バッファで管理する構成の 2 つが考えられる．しかし，どちらの構成が提案手法に適しているか分からない．このため，提案手法に適した送受信バッファの構成を検討する．考えられる 2 つの送受信バッファの構成について図 3.2 と図 3.3 で示し，以下で説明する．

(構成 1) パケットを送信先毎に別々の送受信バッファで管理する構成 (図 3.2)

パケットを送信先別に分類し，対応する送受信バッファで管理する．OS ノード 0 宛のパケットは OS ノード 0 宛のパケットを管理する送受信バッファに格納される．それぞれの OS ノードは，送信時には，送信先のパケットが管理される送受信バッファにパケットを書き込む．受信時には，自身宛のパケットが管理されている送受信バッファからパケットを読み出す．

(構成 2) 異なる送信先のパケットを同一の送受信バッファで管理する構成 (図 3.3)

パケットを送信別に分類せず，同一の送受信バッファで管理する．この構成では，送受信バッファに異なる宛先のパケットが混在するため，受信時に，自身宛のパケットが送受信バッファのどの位置に置かれているか分からない．また，送受信バッファのパケットが書き込まれた順に処理されるとは限らないため，送受信バッファのどの領

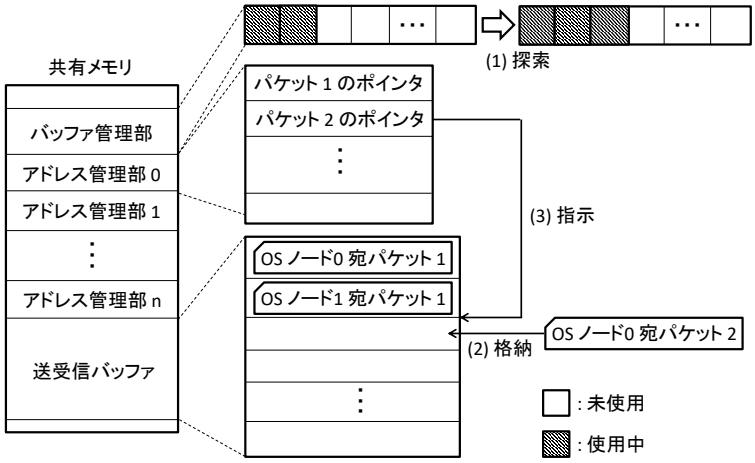

図 3.3 異なる送信先のパケットを同一の送受信バッファで管理する構成

域が書き込み可能か分からない．そこで，送受信バッファとは別に以下の 2 つの領域を共有メモリ上に確保する．

(1) バッファ管理部
送受信バッファの空き状態を管理する領域である．具体的には，送受信バッファの内，どの領域が使用中で，どの領域が未使用かを管理する領域である．

(2) アドレス管理部
送受信バッファに格納されているパケットを指示するポインタを送信先別に分類して管理する領域である．OS ノード 0 宛のパケットを指示するポインタは，OS ノード 0 宛のパケットを指示するポインタを管理するアドレス管理部に格納される．

この構成における送信時の送受信バッファを操作する処理流れを図 3.3 に示し，以下で説明する．

(1) 未使用領域の探索
バッファ管理部から送受信バッファの未使用領域を探す．未使用領域がある場合，その領域のアドレスを取得する．未使用領域がない場合，新たに未使用領域が出来るまで待つ．その後，バッファ管理部を更新し，対応する領域を使用中にする．

(2) パケットの格納
(1) で取得したアドレスにパケットを書き込む．

(3) パケットをポインタで指示
(2) で書き込んだパケットを指示するポインタを送信先のパケットを指示するポイ

表 3.1 構成の利点と欠点

	利点	欠点
構成 1	処理工数が小さい	送受信バッファの利用率が低い
構成 2	送受信バッファの利用率が高い	処理工数が大きい

ンタを管理するアドレス管理部に書き込む．アドレス管理部に空き領域がない場合，新たに空き領域ができるまで待つ．これにより，アドレス管理部に書き込まれているポインタから，パケットを指示できる．

受信時は，送信時の処理と逆の操作を行うことで，送受信バッファからパケットを読み出す．具体的には，まず，アドレス管理部から自身宛のパケットを指示するポインタを取得し，ポインタの格納されていた領域を初期化する．次に，取得したポインタからパケットを読み出す．最後に，バッファ管理部を更新し，パケットが格納されていた領域を未使用領域にする．これにより，送受信バッファからパケットを読み出す．

それぞれの構成における利点と欠点について表 3.1 に示し，以下で比較する．

(比較 1) 送受信バッファ操作の処理工数

（構成 1）では，パケットを送信先別に分類し異なる送受信バッファで管理する．このため，送信時には送信先のパケットが管理される送受信バッファを，受信時には自身宛のパケットが管理される送受信バッファを操作すれば良い．一方で,（構成 2）では，通信時には送受信バッファの他にバッファ管理部とアドレス管理部の操作が必要がある．このため,（構成 1）は（構成 2）よりも送受信バッファ操作の処理工数が小さい．

(比較 2) 送受信バッファの利用率

（構成 1）では，パケットを送信先別に分類し，異なる送受信バッファで管理する．このため，送信先のパケットを管理する送受信バッファに未使用領域がなければそれ以外の送受信バッファに未使用領域があったとしても送信できない．このため，特定の OS ノードに送信が集中する場合，送受信バッファの利用率が低下する．同様に,（構成 2）では，パケットを指すポインタを送信先別に分類し，異なるアドレス管理部で管理するため，送信先のパケットを指すポインタを管理するアドレス管理部に空き領域がなければパケットを送信できない．しかし，アドレス管理部のサイズは送受信バッファと比べて極めて小さい．このため，アドレス管理部のサイズを大きくすることで，送受信バッファで管理できるパケットの量に与える影響を抑えつつ，送受信バッファの

利用率を上げることが出来る．以上から,（構成 2）は（構成 1）よりも送受信バッファの利用率が高い．

（比較 1）から,（構成 1）は（構成 2）よりも送受信バッファの操作における処理工数が小さい．このため,（構成 1）ではより高速な通信が可能となると考えられる．しかし,（構成 2）におけるバッファ管理部とアドレス管理部の操作にかかる時間は送受信バッファの操作に比べて非常に小さい．このため，各構成間での送受信バッファの操作にかかる時間の差は極めて小さいと言える．

（比較 2）から，特定の OS ノードに送信が集中する場合においても,（構成 2）は（構成 1）よりも送受信バッファの利用率が高い．このため,（構成 2）では，特定の OS ノードに送信が集中することによる通信性能への影響が小さい．

以上から，本稿では，送受信バッファの構成に（構成 2）を採用する．

3.4.2 排他制御すべき操作の検討

排他制御によりロックを保持する時間が長い場合，ロックを保持した状態でスケジューラにより割り込まれる可能性がある．このため，ロックを獲得するまでの待ち時間が長大化すると考えられる．このため，ロックを保持する時間が最小となるように，排他制御すべき操作を検討する必要がある．

3.4.1 項から，提案手法で操作する共有資源は，バッファ管理部，アドレス管理部，および送受信バッファの 3 つである．よって排他制御すべき操作の候補は，これらの資源を参照/更新する操作である．バッファ管理部とアドレス管理部を参照/更新する操作は，複数の OS ノードが同時に操作することが考えられるため，排他制御すべきである．送受信バッファを参照/更新する操作は，バッファ管理部とアドレス管理部を参照/更新する操作が排他制御されている場合，複数の OS ノードが同時に同一の領域を操作することはない．このため，排他制御しなくてもよい．

以上から，提案手法では，バッファ管理部とアドレス管理部を参照/更新する操作をそれぞれ排他制御する．

第 4 章

実装

4.1 仮想ネットワークインタフェースの処理流れ

提案手法における VNI の処理流れを図 4.1 に示し，以下で説明する．

(1) プロトコルからパケットの受け取り
送信側の VNI は，プロトコルスタックからパケットを受け取る．

(2) バッファ管理部の更新
送信側の VNI は，バッファ管理部からバッファ管理部を更新することで，未使用領域を使用中に変更する．

(3) パケットの書き込み
送信側の VNI は，(2) で探した送受信バッファの未使用領域にパケットを書き込む．

(4) パケットのポインタを格納
送信側の VNI は，送受信バッファのパケットを指示するポインタを送信先のアドレス管理部に格納する．

(5) IPI の送信
送信側の VNI は，受信側の OS ノードが占有するコアに IPI を送信する．

(6) 受信処理の呼び出し
受信側の OS ノードは，IPI を受信すると VNI の受信処理を呼び出す．

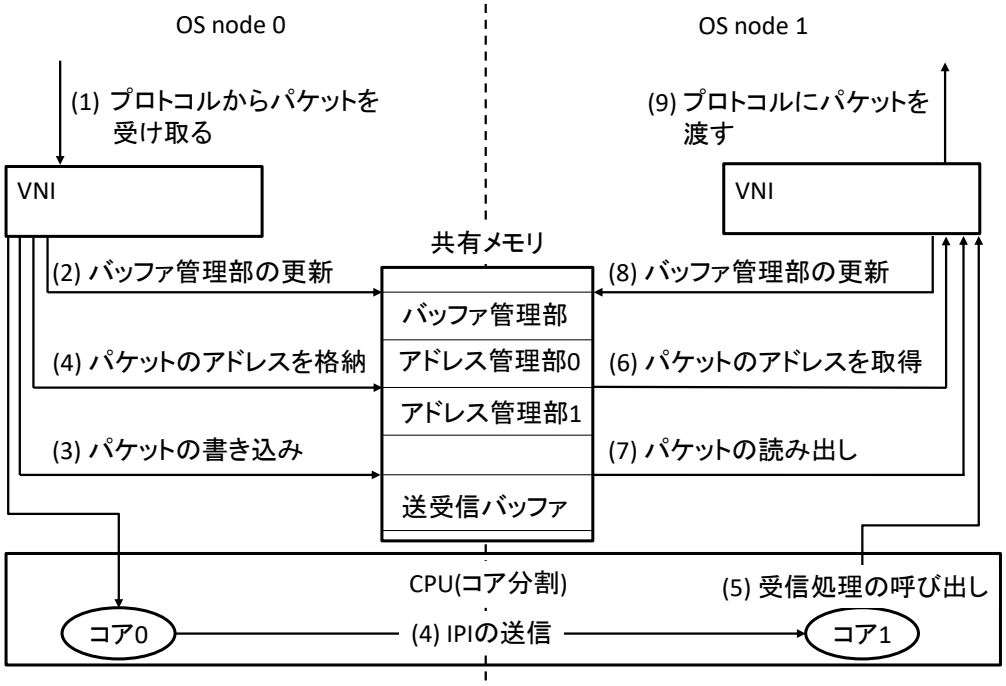

図 4.1 VNI を用いた通信の処理流れ

(7) パケットのポインタを取得
受信側の VNI は，パケットを指示するポインタをアドレス管理部から取得する．

(8) パケットの読み出し
受信側の VNI は，(7) で取得したポインタを用いて送受信バッファからパケットを受け取る．

(9) バッファ管理部の更新
受信側の VNI は，パケット管理部を更新し，パケットが格納されていた領域を未使用領域にする．

(10) プロトコルへパケットを渡す
受信側の VNI は，受信したパケットをプロトコルスタックに渡す．

第 5 章

評価

5.1 評価項目

提案手法について以下の評価項目が考えられる．

(項目 1) アプリケーション改変の有無

提案手法では，既存 AP を用いた場合においても，アプリケーションの改変を必要としない．このため，改変を行っていない既存 AP を用いた通信においても，正しく通信が行えるかを評価する．

(項目 2) 提案手法の実装によるコード量

提案手法では，Linux における提案手法の影響を局所化する必要がある．このため，コード改変量をまとめ，提案手法の影響が局所化できているか評価する．

(項目 3) 性能評価

提案手法による通信は，共有メモリを介して行われるため，NIC ハードウェアを用いた通信手法と比べて高速であると考えられる．しかし，プロトコルスタックの処理が通信速度にどの程度支配的であるか分からないため，その程度は予測しづらい．このため，提案手法の通信性能について評価する．具体的には，提案手法における以下の 3 つの項目について測定し，評価する．

(1) 通信におけるレイテンシ

(2) 単位時間あたりの通信量

(3) パケットロス率

そのうち，本稿では（項目 1）と（項目 2）について述べる．

5.2 アプリケーション改変の有無

提案手法では，既存アプリケーションの改変なしに通信可能である必要がある．このため，提案手法と NIC ハードウェアを用いた通信手法でテストプログラムを走行させ，同様に通信出来ることを示す．走行させるテストプログラムを以下に示す．

(テスト 1) TCP/IP プロトコルを用いた送受信のテスト

送信側の OS ノードは，特定の文字列を TCP/IP プロトコルを用いて受信側の OS ノードに送信する．受信側では，受け取った文字列を標準出力する．受信側で出力された文字列が送信した文字列と一致しているか確認する．

(テスト 2) ベンチマークツールを用いたテスト

Linux 上で動作するベンチマークツールを用いて通信を行い，正しく通信できていることを確認する．ベンチマークツールには，iperf[4] を使用した．iperf は，IP ネットワーク上の通信における最大帯域幅を測定するツールである．

提案手法と既存手法それぞれで（テスト 1）を行った結果，どちらも，送信側で送信した文字列と受信側で出力された文字列が一致した．このため，提案手法と既存手法ともに正しく通信が行われていることを確認した．また，提案手法で（テスト 2）行った結果，正しく通信できており，実効スループットは $802Mbps$ であった．このことから，提案手法では，既存アプリケーションの改変なしに通信可能であり，実行スループットは $802Mbps$ である．

5.3 提案手法の実装によるコード量

提案手法を実装により追加したコード量を明らかにする．なお，本評価では，コード量の評価基準として論理 LOC（Lines Of Code）を使用した．論理 LOC は，ソースコードの総行数から，記号のみの行，空白行，およびコメントのみの行を省いた行数である．論理 LOC はの算出には，CLOC[5] を使用した．CLOC を用いて論理 LOC を算出した結果を表 5.1 に示し，改変内容を以下に示す．

(改変内容 1) 仮想ネットワークインタフェースとするデバイスドライバ (通番 1，2)

提案手法では，OS ノード間通信を Ethernet 互換で行う仮想ネットワークインタフェースとするデバイスドライバとして実現する．このため，このデバイスドライバのソースコードとヘッダファイルを追加した．

表 5.1 提案手法実装におけるコード量

通番	ファイル名	追加行数
1	driver/net/vni/vni.c	456
2	driver/net/vni/vni.h	16
3	driver/net/vni/Makefile	1
4	driver/net/Makefile	1
合計	4 ファイル	474

(改変内容 2) 仮想ネットワークインタフェースのビルド（通番 3，4）

仮想ネットワークインタフェースとするデバイスドライバをビルドするためにこのデバイスドライバをコンパイルする処理を Makefile に記述した．

表 5.1 から，提案手法の実装により追加したコード量は 474 行である．これは，Mint カーネル全体のコード量の約 0.004% と極めて小さい．このため，提案手法の実装は容易であると言える．

第 6 章

おわりに

本稿では Mint における仮想ネットワークインタフェースを用いた OS ノード間通信手法について述べた．まず，Mint オペレーティングシステムにおける既存の OS ノード間通信手法とその問題点について述べた．次に，仮想ネットワークインタフェースによる設計について述べた．その後，設計を実現する上での課題と対処について述べた．そして，仮想ネットワークインタフェースの実装について述べた．最後に，仮想ネットワークインタフェースによる OS ノード間通信の評価結果について述べた．評価の結果，提案手法では，Linux 既存のアプリケーションを改変することなく OS ノード間で通信可能であり，実効スループットが $802Mbps$ であることを示した．また，提案手法の実装のために追加したコード量が 474 行であることを示した．これは，Mint カーネル全体のコード量の約 0.004% であった．したがって，提案手法の実装は容易である．残された課題として，本手法による OS ノード間通信の性能評価がある．具体的には，通信におけるレイテンシ，スループット，およびパケットロス率の評価がある．これにより，通信性能に関わる要因の詳細を解析可能になると考えられる．

謝辞

本研究を進めるにあたり，懇切丁寧なご指導をしていただきました乃村能成准教授に心より感謝の意を表します．また，研究活動において，数々のご指導やご助言を与えていただいた谷口秀夫教授，山内利宏准教授に心から感謝申し上げます．また，日頃の研究活動において，お世話になりました研究室の皆様に感謝いたします．

参考文献

[1] Sugerman, J., Venkitachalam, G. and Lim, B.-H.: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor., *USENIX Annual Technical Conference, General Track*, pp. 1–14 (2001).

[2] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *ACM SIGOPS Operating Systems Review*, Vol. 37, No. 5, pp. 164–177 (2003).

[3] 千崎良太，中原大貴，　牛尾裕，片岡哲也，粟田祐一，乃村能成，谷口秀夫：マルチコアにおいて複数の Linux カーネルを走行させる Mint オペレーティングシステムの設計と評価 (ネットワーク, クラウド及び一般)，電子情報通信学会技術研究報告. CPSY, コンピュータシステム，Vol. 110, No. 278, pp. 29–34 (2010).

[4] iPerf - The ultimate speed test tool for TCP, UDP and SCTP, ikoula.com (online), available from ⟨https://iperf.fr/⟩ (accessed 2018-2-1).

[5] CLOC – Count Lines of Code, sourceforge.net (online), available from ⟨http://cloc.sourceforge.net/⟩ (accessed 2018-2-1).