Vol.2017-OS-139 No.3
2017/3/1

Mint オペレーティングシステムを用いた通信処理の開発支援環境の実現と評価

藤田 将輝[1]　乃村 能成[1]　谷口 秀夫[1]

概要：オペレーティングシステム (OS) 処理の不具合の改修において，割込処理は再現が困難であり，工数の増加を招く．そこで Mint を用いた開発支援手法が提案されている．Mint とは，1 台の計算機上で仮想化を用いず複数の OS を動作させる方式である．提案手法では，開発対象の OS と開発を支援する OS を共存動作させる．また，開発を支援する OS が開発対象の OS のデバイスを擬似することで，割込処理に必要な情報を指定可能であり，処理を再現できる．本稿では，割込処理の 1 つであるパケット受信割込処理に着目する．パケット受信割込処理は，NIC ハードウェアからの割込によって開始し，複雑な通信処理の契機となる処理であり，様々なバグを含みやすい．そこで，パケット受信割込処理を対象に開発支援環境を構築し，通信処理のテストを実施できることを示す．

1. はじめに

OS 機能の複雑化により，OS 開発の工数が増加している [1]．特に，割込処理は発生タイミングや処理すべきデータがハードウェアデバイスの状態に依存するため，不具合の改修時，再現が困難であり，工数の増加を招いている．

Mint オペレーティングシステム (以下，Mint)[2] は，マルチコア CPU を搭載した 1 台の計算機上で仮想化を用いずに複数の Linux を動作させる方式である．この Mint を用いた割込処理の開発支援手法を提案した [3]．提案手法では Mint を用いることで開発対象の OS(以下，開発対象 OS) とは別に開発を支援する OS(以下，開発支援 OS) も動作させる．開発支援 OS が開発対象 OS の特定のハードウェアデバイスを擬似することで制御の困難な割込情報を制御可能にし，開発を支援する．Mint を用いることの利点として，特定のハードウェアの擬似にフルセットの Linux を用いることで開発支援環境のコードを記述しやすいこと，開発支援環境の構築に特殊なハードウェアが必要でないことがある．

ハードウェア割込を契機に開始する処理の 1 つにパケット受信割込処理がある．パケット受信割込処理は NIC ハードウェアの割込を契機に処理を開始する．NIC ハードウェアは年々通信速度が高速化しており [4]，これに対応するソフトウェアの開発が重要視されている．高速な通信に対応するソフトウェアの開発を行うには，高速なデバイスが必要になる．また，パケットサイズや通信速度は NIC ハードウェアによって制限され，これらの情報をデバイスによって調整することは困難である．このため，通信処理のテストにおいて，パケットサイズや通信速度を自由に調整し，テストしたいという要求がある．

そこで本稿では，NIC ハードウェアを擬似対象とした開発支援環境を構築し，構築した環境が通信処理のテストに有用であることを示す．これにより，NIC ハードウェアに制限されるパケットサイズや通信速度を開発支援 OS によって調整可能にする．構築した開発支援環境を用いることで，高速な NIC ハードウェアを想定した通信処理のテストが可能になる．また，NIC ハードウェアを擬似することにより，複数の計算機を用意することなくイーサネットの通信テストが可能になる．このように，1 台の計算機でイーサネットの通信テストが可能になることで，バグの混入しやすいプロトコルスタックにおける OS 処理の開発工数を削減できる．本環境を用いて Linux のテストを行い，以前の Linux に存在した 2 つのバグを発見できたこと，UDP フラッド攻撃の OS に対する影響を明らかにしたことを示す．

2. Mint を用いた開発支援環境

2.1 Mint オペレーティングシステム

Mint オペレーティングシステムは 1 台の計算機上で仮想計算機を用いずに複数の OS を動作させる方式である．Mint では 1 台の計算機上でプロセッサ，メモリ，および

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

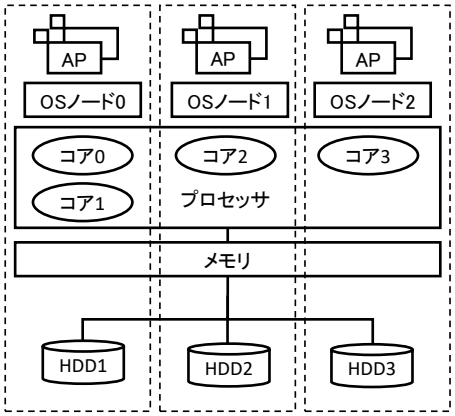

図 1 Mint オペレーティングシステム

デバイスを分割し，各 OS が占有する．Mint の構成例を**図 1** に示し，説明する．本稿では Mint を構成する OS を OS ノードと呼ぶ．Mint では，最初に起動する OS ノードを OS ノード 0 とし，起動順に OS ノード 1，OS ノード 2，… とする．

(1) プロセッサ: コア単位で分割し，各 OS ノードがコアを 1 つ以上占有する．
(2) メモリ: 空間分割し，各 OS ノードが分割領域を占有する．
(3) デバイス: デバイス単位で分割し，各 OS ノードが指定されたデバイスを占有する．

このようにして Mint ではマルチコアプロセッサ CPU のコアを分割し，複数の Linux を同時に走行できる．

2.2 目的

OS の不具合の改修において，割込処理は発生タイミングや処理すべきデータがハードウェアデバイスの状態に依存しているため，再現が困難であり，開発工数の増加を招いている．割込の発生タイミングや処理すべきデータをソフトウェアにより制御できれば，割込処理の再現が容易になり，開発工数を削減できる．そこで，Mint を用いた開発支援環境を構築した [3]．構築した開発支援環境は NIC ハードウェアを擬似対象としている．NIC ハードウェアの性能は年々向上しており，通信速度が高速化している．このため，高速化する通信速度に対応させるための OS の開発が重要になっている．また，通信処理におけるプロトコルスタックの処理は複雑であるため，バグが混入しやすい．そこで，NIC ハードウェアを擬似対象にした開発支援環境を構築することにより，パケット受信割込処理の再現を容易にした．これにより，通信処理におけるプロトコルスタックの開発を支援する．また，本開発支援環境における開発支援 OS が高速な NIC ハードウェアを擬似することで，高速な通信処理の開発を可能にする．

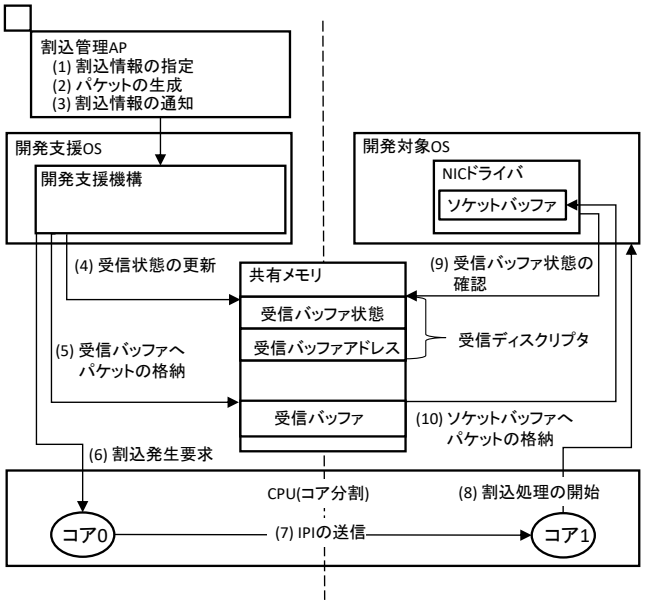

図 2 開発支援環境の構成と処理流れ

2.3 開発支援環境の構成と処理流れ

本開発支援環境の構成と処理流れを**図 2** に示し，説明する．開発支援環境の構成について以下で説明する．

開発支援 OS

開発対象 OS の NIC ハードウェアを擬似する OS である．指定したパケット受信に関する情報を基に NIC ハードウェアの動作を擬似する．開発支援 OS は以下の要素を持つ．

(A) 割込管理 AP
開発支援 OS 上で動作するアプリケーションである．パケット受信処理に必要な情報を指定する．パケット受信処理における割込情報は，受信パケット数，パケットサイズ，パケットのプロトコル，およびパケット挿入間隔である．パケットのプロトコルの指定において，ヘッダやデータを指定できる．割込管理 AP では，指定された割込情報の受信パケット数，パケットサイズ，およびパケットのプロトコルからパケットを生成する．生成したパケットと指定した情報をシステムコールを用いてカーネルに通知する．これにより，開発支援機構が動作を開始する．

(B) 開発支援機構
開発支援 OS に組み込むカーネルの機能である．NIC ハードウェアのパケット受信機能を擬似する．割込管理 AP が発行するシステムコールにより情報を受けとり，処理を開始する．開発支援 OS と開発対象 OS の両 OS がアクセス可能な共有メモリに配置された受信バッファへパケットを格納し，開発対象 OS に IPI を送信する．これにより，開発対象 OS に割込が発生し，パケット受信割込

処理が開始する．

開発対象 OS

開発対象の OS であり，NIC ドライバで割込処理を行う．開発対象 OS は以下の要素を持つ．

(A) NIC ドライバ

開発対象 OS が保持する NIC ドライバである．パケット受信割込処理を行う．本開発支援環境の適用に関して，主に以下の改変を行う必要がある．

(a) 受信バッファを共有メモリへ配置

パケット受信には共有メモリを使用するため，NIC ドライバの初期化時に受信バッファを共有メモリに配置するよう改変する必要がある．

(b) 受信処理に使用する情報の共有

受信処理を行うにあたって，NIC ハードウェアと NIC ドライバ間で共有すべき情報 (レジスタ等) を共有メモリを用いて共有するよう配置する必要がある．

(c) 割込の通知

開発支援 OS の占有するコアが送信する IPI によって NIC ドライバの割込ハンドラが動作するよう改変する必要がある．

以上の構成を取ることで開発支援 OS が開発対象 OS の NIC ハードウェアを擬似できる．次に本開発支援環境の処理流れを以下に示す．ここで，受信ディスクリプタとは NIC ドライバと NIC ハードウェアで共有する情報であり，受信バッファの状態を示すものである．

(1) パケット数，パケットサイズ，パケット挿入間隔，およびパケットのプロトコルを指定し，割込管理 AP を動作させる．
(2) 割込管理 AP が指定された情報を基にパケットを生成する．
(3) 割込管理 AP が割込情報とパケットの情報をシステムコールを用いて開発支援機構に通知する．
(4) 開発支援機構が通知された情報を基に受信ディスクリプタの受信バッファ状態を更新し，受信状態にする．受信バッファ状態とは受信バッファがパケットを受信しているか否かの状態である．
(5) 開発支援機構が通知された情報を基にパケットを格納する．
(6) 開発支援機構が通知された情報を基に IPI の送信要求を発行する．
(7) IPI の送信要求を受けて，開発支援 OS の占有するコア 0 が開発対象 OS の占有するコア 1 へ IPI を送信する．
(8) コア 1 が IPI を受けて割込処理を開始する．
(9) NIC ドライバが受信ディスクリプタを確認し，対応する受信バッファのアドレスを求める．
(10) 受信バッファからソケットバッファへパケットを複写し，プロトコルスタックへ送信する．

3. 評価

3.1 評価項目

まず，本開発支援環境においてパケットサイズとパケット挿入間隔を本来 NIC ドライバが想定しているハードウェアの制限を超えて指定できることを評価する．これにより，本開発支援環境を用いることで高性能な NIC ハードウェアを想定したテストが可能であることを示す．したがって，性能評価として以下の項目を評価する．

(1) パケットサイズ

本開発支援環境が指定できるパケットサイズの範囲と NIC ハードウェアを用いた環境で実現できるパケットサイズの範囲を比較する．これにより，本開発支援環境が NIC ハードウェアよりも広い範囲のサイズのパケットを処理できることを示す．

(2) パケット挿入間隔

本開発支援環境が実現できるパケット挿入間隔の最小値を測定し，本開発支援環境が NIC ハードウェアを用いた場合よりも高速なパケット受信のテストが可能であることを示す．

次に，実装コスト評価として，本開発支援環境の構築による開発対象 OS の改変がわずかなものであることを評価し，開発対象 OS における開発支援環境の構築の影響が局所化できていることを示す．

最後に，有用性評価として，本開発支援環境が指定できる情報であるパケットサイズ，ヘッダとデータ，およびパケット挿入間隔をそれぞれ用いた 3 つのテストを Linux に実施し，本開発支援環境が OS のテストに有用であることを示す．

3.2 評価環境

本稿における評価環境を**表 1** に示す．

表 1 評価環境

項目名	環境
OS	Mint (Linux 3.0.8 改)
CPU	Intel(R) Core(TM) Core i7-870 @ 2.93GHz 開発支援 OS: 1 コア, 開発対象 OS: 1 コア
メモリ	2 GB 開発支援 OS: 256 MB, 開発対象 OS: 256 MB
NIC ドライバ	RTL8169

3.3 性能評価

3.3.1 パケットサイズ

本開発支援環境が NIC ハードウェアよりも広い範囲のサイズのパケットを処理できることを示すため，NIC ハードウェアと開発支援環境それぞれが実現できる最小と最大

のパケットサイズを比較し，評価する．これにより，本開発支援環境が広い範囲のサイズのパケットを扱う NIC ハードウェアを想定したテストが可能であることを示す．ここでパケットサイズとはフレームのサイズとする．NIC ハードウェアと開発支援環境それぞれが受信できるパケットの最小と最大のサイズを**表 2** に示し，以下で説明する．

表 2 NIC ハードウェアと開発支援環境が実現するパケットサイズ

環境	最小サイズ	最大サイズ
NIC ハードウェア (TCP/UDP)	54/42 (B)	1518 (B)
開発支援環境 (TCP/UDP)	54/42 (B)	16384 (B)

(1) 最小サイズ
パケットはヘッダ部とデータ部に分けられる．最小サイズのパケットはデータサイズが 0 バイトのヘッダのみのパケットである．本開発支援環境では TCP パケットか UDP パケットを指定し，作成できる．このため，それぞれのプロトコルのパケットの最小サイズを求める．最小サイズは TCP/UDP で，それぞれ 54/42 バイトとなる．このサイズはフレームヘッダ，IPv4 ヘッダ，および TCP/UDP ヘッダのサイズを加算したものである．NIC ハードウェアを用いた場合も同等のサイズである．つまり，実現できる最小のパケットサイズは NIC ハードウェアを用いた場合と開発支援環境を用いた場合で同じである．

(2) 最大サイズ
NIC ハードウェアを用いた場合の通信では，最大で 1518 バイトのパケットを受信できる．これは MTU として定義されているサイズである．開発支援環境で作成できる最大パケットサイズは NIC ドライバの受信バッファエントリのサイズである．評価対象の NIC ドライバである RTL8169 では受信バッファの 1 エントリが 16384 バイト (16KB) である．つまり，RTL8169 のドライバでは 16KB までのパケット受信を想定して作成されている．したがって，実現できる最大サイズは開発支援環境を用いた場合の方が NIC ハードウェアを用いた場合よりも大きく，NIC ドライバの性能限界をテスト可能である．

以上の結果から，本開発支援環境は NIC ハードウェアを用いた場合よりも広い範囲のサイズのパケット受信を実現できることが分かる．これにより，本開発支援環境は広い範囲のパケットサイズを扱う NIC ハードウェアを想定したテストが可能である．

3.3.2 パケット挿入間隔

本開発支援環境が NIC ハードウェアを用いた通信よりも高速な通信を実現できることを示すため，NIC ハードウェアが実現できる通信速度と本開発支援環境が実現できる通信速度を比較し，評価する．本開発支援環境が実現可能な通信速度を求めるため，開発支援機構のパケット挿入間隔の最小値を求める．パケット挿入間隔とは開発支援 OS において開発支援機構が連続でパケット挿入処理を行おうとした際にパケット挿入処理を開始してから次のパケット挿入処理が開始するまでの時間である．したがってパケット挿入間隔の最小値を求めるには 1 回のパケット挿入処理を考えれば良い．パケット挿入処理は開発支援機構が行う NIC ハードウェアのパケット受信機能であり，以下の処理を行う．

(1) 受信ディスクリプタを更新し，受信状態にする．
(2) パケットを受信バッファに配置する．
(3) IPI を送信し，開発対象 OS に割込を通知する．

パケット挿入処理にはメモリコピー処理が含まれるため，パケットのサイズによってパケット挿入処理時間が異なる．したがって以下の 3 つのサイズでパケット挿入処理時間を測定する．

(1) 1518 バイト (MTU のサイズ)
(2) 8192 バイト (受信バッファの 1 エントリの半分)
(3) 16384 バイト (受信バッファの 1 エントリのサイズ)

結果を**表 3** に示す．結果から，本開発支援環境は NIC ハードウェアを用いた場合よりも高速なパケット受信が可能であることが分かる．したがって，本開発支援環境は高速な NIC ハードウェアを想定したテストが可能である．

表 3 各パケットサイズにおけるパケット挿入処理時間

パケットサイズ (B)	処理時間 (μ s)	通信速度 (Gbps)
(NIC)1518	-	0.9
1518	0.205	55.2
8192	1.462	41.7
16384	3.664	33.3

3.4 実装コスト評価

本開発支援環境の構築による開発対象 OS の改変がわずかであることを示すため，開発対象 OS の改変箇所と改変量を評価する．これにより，開発支援環境の構築による開発対象処理への影響を局所化できるていることを示す．

開発支援環境の構築によって改変するのは NIC ドライバのみである．また，主な改変は NIC ドライバのバッファの配置先といった，どのような NIC ドライバでも定数として定義できる簡単な箇所であり，容易に特定可能な改変箇所である．具体的な NIC ドライバへの改変は以下の 6 箇所である．

(1) 各種ハードウェアレジスタのマッピング先
開発支援 OS が NIC ハードウェアの擬似をするため，レジスタを用いた情報は開発支援 OS が参照できる必要がある．そこで，各種ハードウェアレジスタのマッピング先を変更し，開発支援 OS と開発対象 OS の共有メモリとする．

(2) 受信バッファの確保先
開発支援 OS が受信バッファへパケットを格納する必要があるため，受信バッファの確保先を変更し，開発支援 OS と開発対象 OS の共有メモリとする．

(3) 受信ディスクリプタの配置先
開発支援 OS が受信ディスクリプタを操作する必要があるため，受信ディスクリプタの配置先を変更し，開発支援 OS と開発対象 OS の共有メモリとする．

(4) 割込の通知
開発支援 OS が IPI を用いて割込を通知するため，IPI により開発対象 OS の NIC ドライバの割込ハンドラが動作するよう改変を加える．

(5) レジスタ操作
NIC ハードウェアにはパケット受信時のステータスを示すレジスタがある．このステータスは割込発生時に確認され，該当 NIC ハードウェアが発行した割込かを確かめる．本開発支援環境においては IPI によって割込が発生し，ハードウェア的なエラーは無いことを前提としているため，正しいステータスを取得させる．

(6) NIC ハードウェアの停止
本開発支援環境において，NIC ハードウェアは用いない．しかし，ネットワークインタフェースの起動時にハードウェアが存在しない場合，NIC ドライバがカーネルに組み込まれない．そこで，初期化処理時にのみ NIC ハードウェアを使用し，初期化処理の末尾で NIC ハードウェアを停止する．

次に，開発支援環境の実装における NIC ドライバの改変量を**表 4** に示す．表 4 より，合計の追加行数が 8 行，削除行数が 7 行である．RTL8169 のコード行数である約 5,500 行と比較してわずかな改変であると言える．これらから，開発支援環境の構築による開発対象処理への影響を局所化できていると言える．

表 4 NIC ドライバの改変行数

内容	追加	削除
各種ハードウェアレジスタのマッピング先	1	1
受信バッファの確保先	1	2
受信ディスクリプタの配置先	2	2
割込の通知	1	0
レジスタ操作	2	2
NIC ハードウェアの停止	1	0
合計	8	7

3.5 有用性評価

3.5.1 テスト項目

本開発支援環境が OS のテストに有用であることを示すため，実際に Linux にテストを実施する．本開発支援環境では，実際はハードウェアにより制限される以下の情報を自由に設定できる．

(1) パケットサイズ
(2) パケットのヘッダとデータ
(3) パケット挿入間隔

そこで，それぞれの情報を用いて以下の 3 つのテストを実施する．

(テスト 1) パケットサイズを徐々に増加させるテスト
サイズを最小値から最大値まで 1 バイト単位で増加させたパケットを処理させ，挙動を確認するテストである．カーネルパニック等を発生しないことの確認や許容すべきサイズのパケットを許容し，破棄すべきサイズのパケットを破棄しているかを確認する．

(テスト 2) ランダムな値のパケットを処理させるテスト
プロトコルスタックの処理においてカーネルは，どのようなパケットを受信しても処理を停止せず正しく処理しなければならない．そこで，ヘッダとデータをランダムな値にしたパケットを大量に生成し，開発対象 OS に処理させることでカーネルが正しくパケットを処理していることを確認する．

(テスト 3) 高速に大量のパケットを受信させるテスト
本開発支援環境ではハードウェアの制限を超えた高速なパケット受信が可能である．そこで，高速に大量のパケットを受信させ，この際の OS の挙動を確認する．このテストは UDP フラッド攻撃のストレステストとして実施できる．

3.5.2 テスト対象バグ

(テスト 1) と (テスト 2) はバグの発見を目的としたテストである．このため，開発対象 OS のカーネルに以前の Linux に存在したバグを挿入し，(テスト 1) と (テスト 2) を実施する．これにより，各テストでバグを発見できることを示す．開発対象 OS のカーネルに挿入するバグを以下に示す．

(バグ 1) 特定のサイズのパケットを破棄してしまうバグ
(バグ 2) 特定のヘッダのパケットを受信するとカーネルパニックを発生するバグ

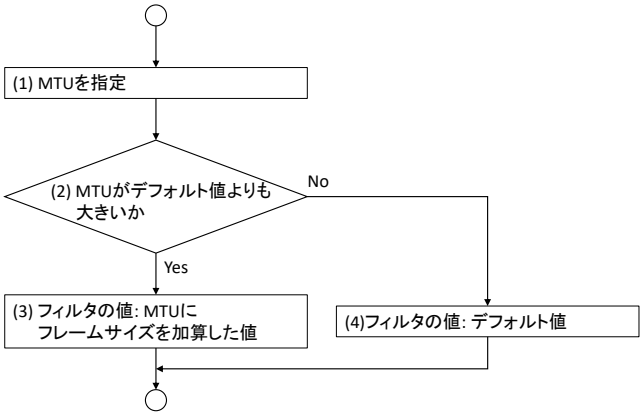

図 3 バグを含んだハードウェアフィルタ設定の処理流れ

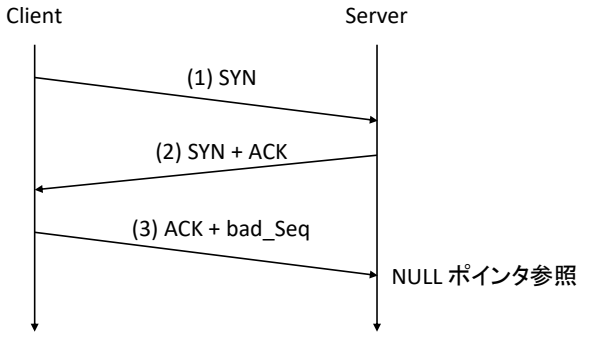

図 4 TCP の通信確立においてバグが発生するまでの処理流れ

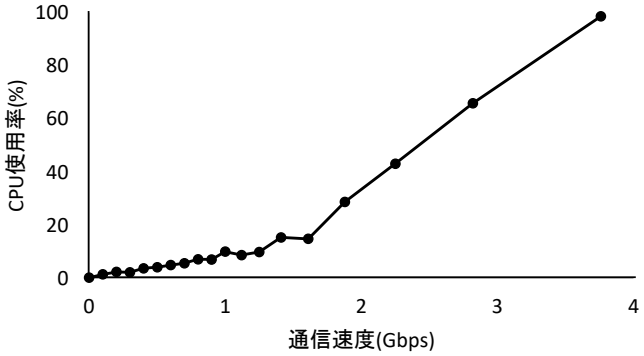

図 5 UDP フラッド攻撃時の CPU 使用率

(バグ 1) は，Linux 2.6.31 の NIC ドライバ (RTL8169) に存在したバグである．RTL8169 ではハードウェアフィルタによって受信できる最大のパケットサイズを設定している．ハードウェアフィルタは MTU を基に算出され，ハードウェアフィルタを超えるサイズのパケットはハードウェアによって破棄される．ハードウェアフィルタはフレームサイズとして定義され，MTU は IP パケットサイズとして定義されるものである．本バグはハードウェアフィルタの設定に誤りがあり，パケットフィルタが IP パケットサイズとして定義されてしまうことにより，許容されるはずのパケットが破棄されてしまうバグである．バグが発生する際のハードウェアフィルタ設定の処理流れについて，**図 3** に示し，以下で説明する．

(1) MTU を設定し，ネットワークインタフェースを起動する．
(2) 設定した MTU の値がデフォルト値よりも大きければ (3) へ遷移する．デフォルト値以下であれば (4) へ遷移する．
(3) フィルタの値として，MTU の値にフレームヘッダ等の値が加算された値が設定される．
(4) フィルタの値としてデフォルト値が設定される．

上記の処理流れにおいて，MTU を 1500 と設定し，デフォルト値も 1500 であった場合，MTU とデフォルト値が一致しているため，ハードウェアフィルタは 1500 と設定される．つまり，1501 バイト以上のパケットはハードウェアによって破棄されることになる．しかし，MTU を 1500 として設定した場合，正しくは 1522 バイトまでのパケットを受信できるはずであり，1501 バイトから 1522 バイトのパケットが本来許容されるべきであるのにも関わらず破棄されてしまう．

(バグ 2) は， Linux 2.6.27-rc2 の TCP ハンドラに存在したバグである．本バグは文献 [5] で紹介されていたバグである．TCP を用いて通信を行う際，通信の確立に 3 回の通信が必要である．本バグが発生するまでの処理流れについて**図 4** に示し，以下で説明する．

(1) クライアントからサーバへ SYN パケットが送信される．
(2) サーバからクライアントへ SYN パケットと ACK パケットが送信される．
(3) クライアントからサーバへ ACK パケットが送信される．この際，ACK パケットに含まれるシーケンス番号が期待したものと異なっていた場合，NULL ポインタ参照が発生し，カーネルパニックを引き起こす．

3.5.3 テスト結果

それぞれのテストの結果について以下に示す．

(テスト 1) パケットサイズを徐々に増加させるテスト
特定サイズのパケットにおいて許容すべきサイズのパケットを破棄していることを確認したため，(バグ 1) を発見できた．

(テスト 2) ランダムな値のパケットを処理させるテスト
10 回程テストを実施したところ，全ての試行においてテストを実行してから 1 秒以内にカーネルパニックが発生することを確認し，(バグ 2) を発見できた．

(テスト 3) 高速に大量のパケットを受信させるテスト
パケット挿入間隔を調整し，4Gbps までの通信速度の UDP フラッド攻撃を再現した．この際の CPU 使用率を測定し，UDP フラッド攻撃の影響を調査した．結果を**図 5** に示す．結果から 0Gbps〜2Gbps までは CPU 使用率がそれほど上昇していないことが分かる．また，2Gbps〜4Gbps の範囲の攻撃では CPU 使用率が大きく上昇していることが分かる．これらから本評価環境では 2Gbps 以上の攻撃を受けた際，性能に大きく影響が出ることが分かる．さらに，本テストから本評価環境において NIC ハードウェアを 2Gbps まで高速化させても CPU 使用率が大きく上昇し，他の処理に大きな影響が出てしまうことが分かる．また，本開発支援環境はその原因の調査に有用であると考えられる．これは本評価で使用した計算機の NIC ハードウェア (1Gbps) ではテストが不可能であるため，本開発支援環境を用いた場合にのみ可能なテストであると言える．

4. 関連研究

4.1 プロトコルスタックの開発支援を目的とした研究

ネットワークプロトコルスタックの開発を目的とした研究として packetdrill[6] がある．packetdrill はユーザアプリケーションとして実装されたツールであり，パケットの送受信やシステムコールをスクリプトとして記述し，テストを実行する．スクリプトとして記述することにより，処理の再現が容易になり，バグの再現にかかる工数を削減できる．packetdrill を用いたテストはほとんどが 1 秒未満で完了するため，迅速に繰り返しテストできる．さらに，仮想的なネットワークデバイスを使用することにより，1 台の計算機でプロトコルスタックの開発が可能である．しかし，NIC ドライバを開発対象に含めようとすると 2 台以上の計算機が必要になる．本開発支援環境では Mint を用いて一方の OS が他方の OS の NIC ハードウェアを擬似することにより 1 台の計算機上で NIC ドライバを含めた開発を行える．

4.2 高速なパケット受信を目的とした研究

フラッド攻撃のテストのために非常に高速なパケット受信を生成する研究として文献 [7] がある．文献 [7] では FPGA を用いて高速な SYN フラッド攻撃を行うチップを作成し，高速な攻撃を再現している．文献 [7] ではソフトウェアベースのツールである Hping3[8] と文献 [7] の提案手法の速度を比較している．結果，文献 [7] の提案手法は Hping3 よりも高速なパケット受信を実現している．文献 [7] では SYN フラッドを対象としていたが，FPGA を用いて設計を行なっているため，チップの再構成が可能であり，その他の様々な攻撃に応用することが可能と記述されている．

本研究の提案手法は文献 [7] と同等以上に高速な攻撃を再現可能である．これは，1 つのコアを NIC ハードウェアのパケット受信処理のみに割り当てることで高速なパケット受信を実現しているためだと考えられる．また，開発支援 OS と開発対象 OS それぞれが占有するコアが同一の CPU 上に存在するため，バスを通じた距離が短く，高速な割込通知を実現できていると考えられる．さらに，本研究の提案手法は特殊なハードウェアを必要としない．本研究の提案手法は，マルチコアプロセッサを搭載した計算機であれば構築可能である．

4.3 仮想化を想定した高速な通信手法

仮想化を想定した高速な通信手法として，SR-IOV[9] がある．SR-IOV はハードウェアレベルの仮想化支援機能であり，ゲスト OS がハイパーバイザの仲介なしで I/O デバイスを使用できる．ハイパーバイザの仲介によるオーバヘッドを削減することで高速な通信が可能になる．しかし，仮想マシンを使用しない場合と比較して 15 %から 30 %の性能低下があり，実機ほどの性能を実現できない [10]．また，SR-IOV は NIC ハードウェアを用いて通信のテストを行うことになるため，NIC ハードウェア以上の速度は実現できない．本研究の開発支援環境は Mint を用いることで仮想化を用いずに複数 OS を動作し，一方の OS がメモリに直接パケットを配置する．したがって，実機と同等以上の通信速度が実現できる．

5. おわりに

NIC ハードウェアを擬似対象とした開発支援環境とその評価について示した．Mint を用いることで，1 台の計算機上で開発対象 OS とは別に独立した開発支援 OS も動作できる．開発支援 OS は，開発対象 OS の NIC ハードウェアの動作を擬似し，実際の NIC ハードウェアでは制限されるパケットサイズや通信速度をハードウェアの制限を超えて設定できる．これにより，高速に大量のパケットを受信でき，高速な NIC ハードウェアのシミュレートが可能になる．また，パケットの受信処理を行えるため，処理が複雑でバグの混入しやすいプロトコルスタックの処理の開発も可能になる．

本開発支援環境の性能として，最大で 16KB のサイズのパケットを受信したことと同等の動作を擬似できることを示した．また，パケットサイズが 1518 バイトの場合，約 55Gbps の通信速度でパケットを受信したことと同等の動作を擬似できることを示した．これらにより，性能評価で示した範囲でパケットサイズと通信速度を調整することで，任意の性能の NIC ハードウェアを想定した開発が可能になる．

開発支援環境の構築に必要な開発対象 OS の改変箇所はわずかであり，改変量は，追加行が 8 行，削除行が 7 行である．NIC ドライバのコードの規模で考えた場合，改変量はごくわずかであり，開発支援環境の構築による開発対象 OS への影響を局所化できていることを示した．

本開発支援環境のテストへの有用性を示すため，Linux にバグを挿入し，テストを行った．このテストにより，以前の Linux に存在したバグを 2 つ発見できた．また，高速なパケット受信を利用し，UDP フラッド攻撃の再現を行い，UDP フラッド攻撃が計算機に与える影響について調査した．これらにより，本開発支援環境が OS の通信処理のテストに有用であることを示した．

参考文献

[1] Chou, A., Yang, J., Chelf, B., Hallem, S. and Engler, D.: An empirical study of operating systems errors, *ACM SIGOPS Operating Systems Review*, pp. 73–88 (2001).
[2] 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，電子情報通信学会技術研究報告書，Vol. 110, No. 278, pp. 29–34 (2010).
[3] 藤田将輝，乃村能成，谷口秀夫：Mint オペレーティングシステムを用いた NIC ドライバの開発支援手法の実現，情報処理学会研究報告，Vol. 2016-OS-136, No. 8, pp. 1–8 (2016).
[4] Belson, D.: Q2 2016 State of the Internet - Connectivity Report, akamai (online), available from ⟨https://www.akamai.com/jp/ja/multimedia/documents/state-of-the-internet/akamai-state-of-the-internet-connectivity-report-q2-2016.pdf⟩ (accessed 2017-01-08).
[5] 宮原俊介，吉村　剛，山田浩史，河野健二：仮想マシンモニタを用いた割込み処理のデバッグ手法，情報処理学会研究報告，Vol. 2013-OS-124, No. 6, pp. 1–8 (2013).
[6] Cardwell, N., Cheng, Y., Brakmo, L., Mathis, M., Raghavan, B., Dukkipati, N., Chu, H.-k. J., Terzis, A. and Herbert, T.: packetdrill: Scriptable network stack testing, from sockets to packets, *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pp. 213–218 (2013).
[7] Ghanti, S. R. and Naik, G.: Design of System on Chip for Generating SYN Flood Attack to Test the Performance of the Security System, *International Journal of Computer Applications*, Vol. 2015-IJCA-122, No. 7, pp. 14–17 (2015).
[8] Sanfilippo, S.: Hping3, (online), available from ⟨http://www.hping.org/hping3.html⟩ (accessed 2017-01-08).
[9] : SR-IOV, PCI Special Interest Group (online), available from ⟨http://pcisig.com/⟩ (accessed 2017-01-25).
[10] Musleh, M., Pai, V., Walters, J. P., Younge, A. and Crago, S.: Bridging the virtualization performance gap for HPC using SR-IOV for InfiniBand, *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pp. 627–635 (2014).