修 士 論 文

題 目

Mint オペレーティングシステムを用いたカーネル開発支援環境の実現

指導教員

報 告 者

藤田 将輝

岡山大学大学院 自然科学研究科 電子情報システム工学専攻

平成 29 年 2 月 3 日 提出

要約

オペレーティングシステム (OS) 機能を実現する処理の中でも割込処理は発生タイミングや処理すべきデータがデバイスの状態に依存する．このため，不具合の改修時，処理の再現が困難であり開発工数の増加を招く．割込処理の開発を支援する研究として Mint を用いた開発支援手法を提案した．Mint とは 1 台の計算機上で仮想化を用いずに複数の OS を動作させる方式である．提案手法では Mint を用いて 1 台の計算機上に開発対象の OS と開発を支援する OS が共存動作する．開発を支援する OS は開発対象の OS の特定のハードウェアデバイスを擬似する．これにより，デバイスに依存している割込のタイミングや処理すべきデータを制御可能にする．割込のタイミングやデータを制御可能にすることで割込処理を再現可能にし，開発工数を削減する．

デバイスからの割込を契機として動作を開始する処理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 ハードウェアからの割込によって動作し，通信処理の契機となる処理である．パケット受信割込処理を契機とする通信処理はプロトコルスタック部の処理であり，実装が複雑である．したがって，プロトコルスタックにはバグが多く，テストが重要視されている．

そこで，本研究では NIC ハードウェアを擬似対象として開発支援環境を構築し，カーネルのテストを実施した．これにより，本開発支援環境が最大で 16384 バイト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，最速で約 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．開発支援環境の構築において，開発対象 OS の NIC ドライバには，8 行追加し，7 行削除した．NIC ドライバのコード規模で考えた場合，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．さらに，開発支援環境を用いて NIC ドライバの通信性能をテストした結果，本評価環境であれば，最速で約 30.6Gbps を実現できることを示した．本開発支援環境を用いて，実際に Linux カーネルをテストしたところ，以前の Linux に存在したバグを 2 つ発見でき，UDP フラッド攻撃が OS に与える影響を明らかにできた．

目次

図 目 次

表 目 次

第 1 章

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，開発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) を共存動作させる．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 を使用でき，コードを記述しやすいこと，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割込処理の 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 ハードウェアからの割込によって開始する．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 [5]．高速な通信に対応するソフトウェアの開発には，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，本論文では，NIC ハードウェアを擬似対象にすることで，パケットサイズや通信速度を自由に調整できる環境を構築し，幅広い性能の NIC ハードウェアを想定したテストが実施できることを示す．また，NIC ハードウェアを擬似することで，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．1 台の計算機で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カーネル処理の開発工数を削減できると考えられる．本開発支援環境を用いてカーネルにおける NIC ドライバとプロトコルスタックのテストを行い，本開発支援環境の有用性を示す．

第 2 章

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 を動作できる方式である．Mint では 1 台の計算機上でプロセッサ，メモリ，およびデバイスを分割し，各 OS が占有する．Mint の構成例を図 2.1 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 の多機能化により，OS 開発に多くの工数が必要になっている．OS 機能を実現する処理の中でも割込処理は割込の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，不具合の改修時，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

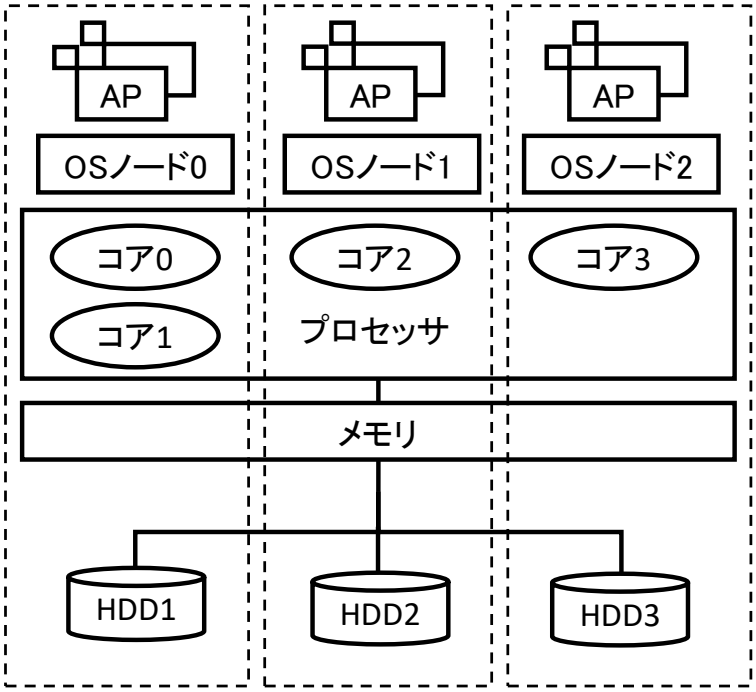

図 2.1 Mint オペレーティングシステム

理の再現が容易になり，開発工数を削減できると考えられる．そこで，Mint を用いた開発支援環境を構築した [3]．Mint を用いた開発支援環境は，Mint を用いることで，開発対象 OS とは別に開発支援 OS が共存動作する．開発支援 OS は開発対象 OS の特定のハードウェアデバイスを擬似し，割込の発生タイミングや処理すべきデータを制御することで，再現の困難な割込処理の再現を容易にする．これにより，開発を支援する．

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 2.2 に示し，説明する．本開発支援環境は開発支援 OS と開発対象 OS により構成される．それぞれについて以下で説明する．

開発支援 OS

開発対象 OS のハードウェアデバイスを擬似する OS である．割込に関する情報を指定可能であり，指定した情報に基づいて開発対象 OS に割込を発生させる．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 上で動作するユーザアプリケーションである．本アプリケーション

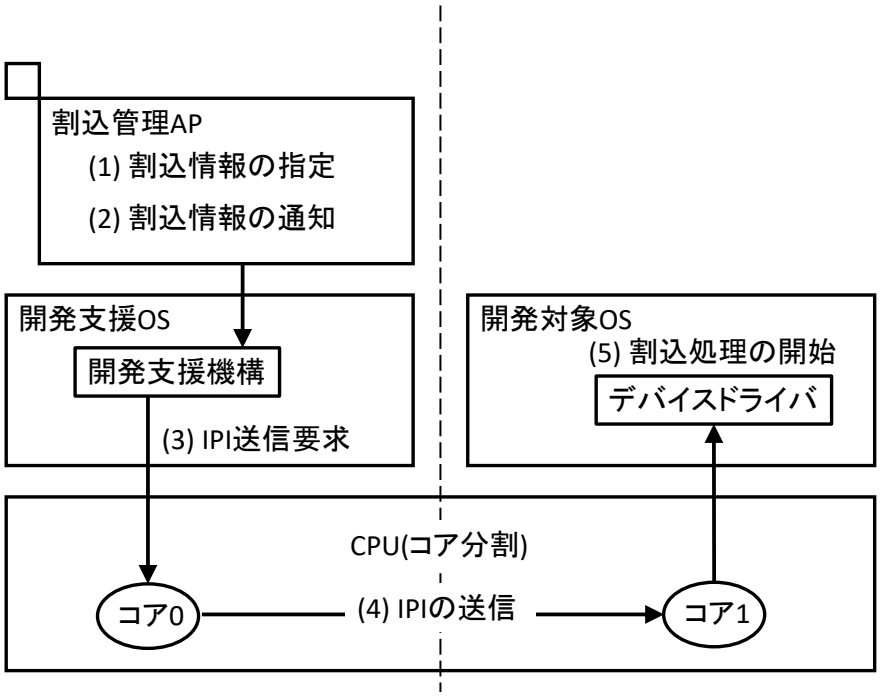

図 2.2 開発支援環境の構成と処理流れ

で割込のタイミングや処理すべきデータといった情報を指定する．指定した情報はシステムコールを用いてカーネルに通知する．

(B) 開発支援機構

開発支援 OS のカーネルに組み込む処理であり，割込管理 AP からのシステムコールによって動作する．特定のハードウェアデバイスを擬似する主体となる処理部である．特定のハードウェアデバイス固有の処理を行う．また，特定のハードウェアデバイスが発行する割込の代わりにコア間割込である IPI を用いて開発対象 OS に割込を発生させる．

開発対象 OS

開発対象の OS であり，割込処理を行う OS である．本開発支援環境では開発支援 OS が特定のハードウェアデバイスを擬似するため，擬似するハードウェアデバイスのドライバをわずかに改変する必要がある．開発対象 OS は以下の要素を持つ．

(A) デバイスドライバ

開発対象 OS で動作するデバイスドライバであり，開発支援 OS が擬似するハードウェアデバイスに対応するデバイスドライバである．擬似するデバイスのレジ

スタ操作や割込ハンドラの動作に関して，開発支援環境に対応させるため，わずかに改変を加える必要がある．

以上の構成を取ることで開発支援 OS が開発対象 OS の特定のデバイスを擬似できる．次に本開発支援環境の処理流れを以下に示す．

(1) 割込管理 AP により，割込情報を指定する．

(2) 割込情報をシステムコールにより通知し，開発支援機構を動作させる．

(3) 開発支援機構が受け取った情報を基にデバイス固有の処理を行い，IPI の送信要求を発行する．

(4) 開発支援 OS の占有するコア 0 が IPI の発行要求を受け取ると，開発対象 OS の占有するコア 1 へ IPI を送信する．

(5) コア 1 が IPI を受信すると開発対象 OS で割込処理が始まり，対応するデバイスドライバの割込ハンドラが動作する．

第 3 章

NIC ハードウェアを擬似対象とした開発支援環境

3.1 目的

NIC ハードウェアの性能向上により，年々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，プロトコルスタックにおける処理は複雑であるため，バグが混入しやすい．パケットの受信は NIC ハードウェアによるパケット受信割込によって開始される．不具合の改修において，ハードウェア割込は発生タイミングがハードウェアの状態に依存しており，非同期に発生するため，再現が困難である．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，パケット受信割込の再現を容易にする．これにより，プロトコルスタックにおけるカーネルの処理の開発を支援する．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パケット処理の開発を可能にする．

3.2 設計方針

本開発支援環境における設計方針を以下に示す．

(方針 1) 開発支援 OS が NIC ハードウェアを擬似する．

本開発支援環境の開発対象はパケットの受信処理である．このため，開発支援 OS はパケット受信に関わるハードウェアデバイスである NIC ハードウェアを擬似する．これにより，パケット受信に関係する情報を制御可能にする．

(方針 2) パケット受信には共有メモリを使用する．

本開発支援環境では開発支援 OS で作成したパケットを開発対象 OS が処理するという構成を取る．そこで，開発支援 OS から開発対象 OS へパケットを受け渡す必要がある．Mint には全ての OS がアクセス可能な共有メモリ空間が存在する．この共有メモリを使用することでパケットの受信を行う．

(方針 3) 開発対象 OS の改変を抑える．

開発支援 OS が NIC ハードウェアを擬似するにあたって，開発対象 OS を改変する必要がある．開発対象の改変範囲が多岐にわたると開発対象処理に大きな影響を与えてしまう．そこで，開発対象 OS の改変は少量にとどめ，影響を局所化する．

(方針 4) パケット受信に関わる情報を自由に設定できる．

NIC ハードウェアを使用した開発では，様々な情報がハードウェアにより制限される．これらの情報を制限なく設定できれば幅広い性能の NIC ハードウェアを想定したテストが可能になる．そこで，本開発支援環境では以下の情報を自由に設定可能にする．

(1) パケットサイズ

NIC ハードウェアを用いた場合，ハードウェア性能の影響により，パケットサイズが制限される．大きなデータを扱う際にはデータを分割し，分割したデータをやり取りする．近年ではジャンボフレームと呼ばれる大きなサイズのパケットを用いて通信回数を削減する手法がある．このため，大きなサイズのパケット通信の開発支援が必要であると考えられる．そこで，本開発支援環境では自由にパケットサイズを指定可能にする．

(2) パケットヘッダとデータ

OS はパケットがどのようなヘッダやデータであっても処理を停止させてはならない．Linux では過去に，プロトコルスタックにおける処理で予期しないデータを受信した際にカーネルパニックを発生するバグが存在した．このようなバグは開発段階で取り除かれるべきである．このようなバグが残っていたのはパケットのヘッダやデータに関するテストが不十分であったためだと考えられる．そこで，本開発支援環境では自由にパケットのヘッダとデータを指定可能にする．

(3) パケット挿入間隔

パケットの挿入間隔の限界はハードウェアに依存する．高速な NIC ハードウェアに対応するドライバの開発を行う場合，高速な NIC ハードウェアを用意しなければならない．また，テストできるのは現存する NIC ハードウェアの速度までにと

どまってしまう．そこで，本開発支援環境では自由にパケットの挿入間隔を指定可能にし，高速な NIC ハードウェアのシミュレートを可能にする．

3.3 設計

3.3.1 概要

本開発支援環境では NIC のパケット受信割込処理を開発対象とする．開発支援 OS が開発対象 OS の NIC ハードウェアを擬似することでパケット受信割込処理で必要な情報を指定可能にする．割込管理 AP において割込処理に必要な情報を指定し，システムコールによりこれらの情報をカーネルに通知する．割込管理 AP が発行したシステムコールによって開発支援機構が動作し，NIC ハードウェアの動作を擬似する．開発支援機構は共有メモリにある NIC の受信バッファへパケットを格納する．NIC ハードウェアの割込については IPI を用いて擬似する．開発対象 OS の占有するコアが IPI を受信することにより，パケット受信割込処理が開始する．

3.3.2 各要素の設計

割込管理 AP，開発支援機構，および NIC ドライバの設計について以下で説明する．

(1) 割込管理 AP

NIC のパケット受信処理に必要な情報を指定する．NIC の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．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．

(2) 開発支援機構

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．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．

(3) NIC ドライバ

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 (方針 2) ことから 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

パケット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) を共有メモリに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．

3.4 実装

3.4.1 概要

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 Linux 3.0.8 に構築した．また，改変を加えた NIC ドライバは RTL8169 である．RTL8169 はパケット受信処理において，受信ディスクリプタと呼ばれるデータ構造を参照することで，受信バッファにパケットを受信していることを確認する．この受信ディスクリプタを未受信状態から受信状態に変更するのは NIC ハードウェアである．この動作を開発支援 OS が再現するには開発支援 OS が受信ディスクリプタを参照可能である必要がある．そこで，受信ディスクリプタを開発支援 OS と開発対象 OS の共有メモリに配置する．また，NIC ハードウェアと NIC ドライバはレジスタを用いて情報をやり取りする．この動作を再現するため，上記のようにレジスタのマッピング先を開発支援 OS と開発対象 OS の共有メモリに配置する．以降で割込管理 AP，開発支援機構，および NIC ドライバ (RTL8169) の実装について述べる．

3.4.2 割込管理 AP

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動作させる．パケットのプロコトルとして，TCP と UDP を設定できる．また，ヘッダの各

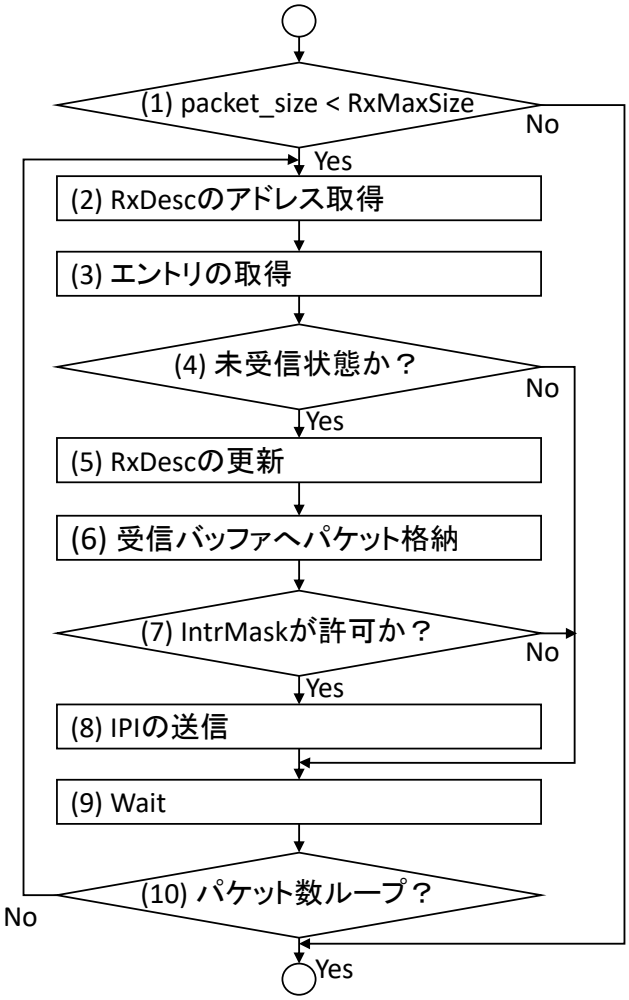

図 3.1 開発支援機構の処理流れ

値をランダムな値として生成できる．これにより，様々なパターンのパケットを生成できる．指定した情報と生成したパケットへのポインタを引数としてシステムコールを用いて開発支援機構を呼び出す．これにより，開発支援機構が動作し，NIC ハードウェアの擬似を行う．

3.4.3 開発支援機構

開発支援機構は NIC ハードウェアの動作を擬似する．開発支援機構の処理流れについて図 3.1 に示し，以下で説明する．ここで RxMaxSize とは NIC ハードウェアが受信できる最大のサイズが格納されたレジスタである．RxDesc とは受信ディスクリプタである．IntrMask とは割込の禁止と許可を示すレジスタである．

(1) 指定したパケットサイズが RxMaxSize よりも小さいかを確認する．つまり，挿入しようとするパケットが NIC ドライバの許容するパケットサイズかどうかを判断する．小さければ (2) へ遷移する．そうでなければ処理を終了する．

(2) 受信ディスクリプタの先頭アドレスを取得する．

(3) 受信ディスクリプタのエントリを取得する．

(4) 求めたエントリの受信ディスクリプタが未受信状態かを確認する．未受信状態であれば (5) へ遷移する．受信状態であれば (9) へ遷移する．

(5) 受信ディスクリプタを更新し，受信状態にする．

(6) 求めたエントリの受信ディスクリプタが示す受信バッファエントリへパケットを格納する．

(7) IntrMask を確認し，割込が許可されていることを確認する．許可されていれば (8) へ遷移する．禁止されていれば (9) へ遷移する．

(8) IPI の送信要求を発行する．

(9) 指定された間隔となるまで Wait する．

(10) 指定されたパケット数ループしたかを確認する．ループ回数と指定したパケット数が一致していれば処理を終了する．していなければ (2) へ遷移する．

3.4.4 NIC ドライバ

開発支援環境を適用するため，NIC ドライバには以下のような改変を行った．

(1) 各種ハードウェアレジスタのマッピング先
NIC ドライバと NIC ハードウェアはハードウェアレジスタを用いて情報をやり取りする．本開発支援環境では開発支援 OS が NIC ハードウェアの擬似をするため，ハードウェア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

NIC ハードウェアはパケットを受信すると受信したパケットを NIC ドライバの管理する受信バッファに格納する．本開発支援環境では開発支援 OS が NIC ハードウェアの擬似をするため，開発支援 OS が受信バッファを参照できる必要がある．そこで，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

NIC ハードウェアはパケットを受信すると受信ディスクリプタを更新し，NIC ドライバにパケットの受信状態やサイズを通知する．また，NIC ドライバはパケットの受信処理を行う際，受信ディスクリプタを参照し，パケットを受信しているか否かを確認する．本開発支援環境では開発支援 OS が NIC ハードウェアの擬似をするため，開発支援 OS が受信ディスクリプタを参照できる必要がある．そこで，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

NIC ハードウェアはパケットを受信バッファに格納したことを開発対象 OS に通知するため，パケット受信割込を発生させる．本開発支援環境では開発支援 OS が NIC ハードウェアを擬似するため，開発支援 OS から開発対象 OS へ割込を発生させる必要がある．そこで，開発支援 OS への 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できるよう改変する必要がある．

(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．このレジスタは割込発生時に確認され，該当ハードウェアが発行した割込かどうかを確かめる．本開発支援環境ではハードウェアのエラーは発生しないことを前提としているため，正しいステータスを常に取得できるものとする．

(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．

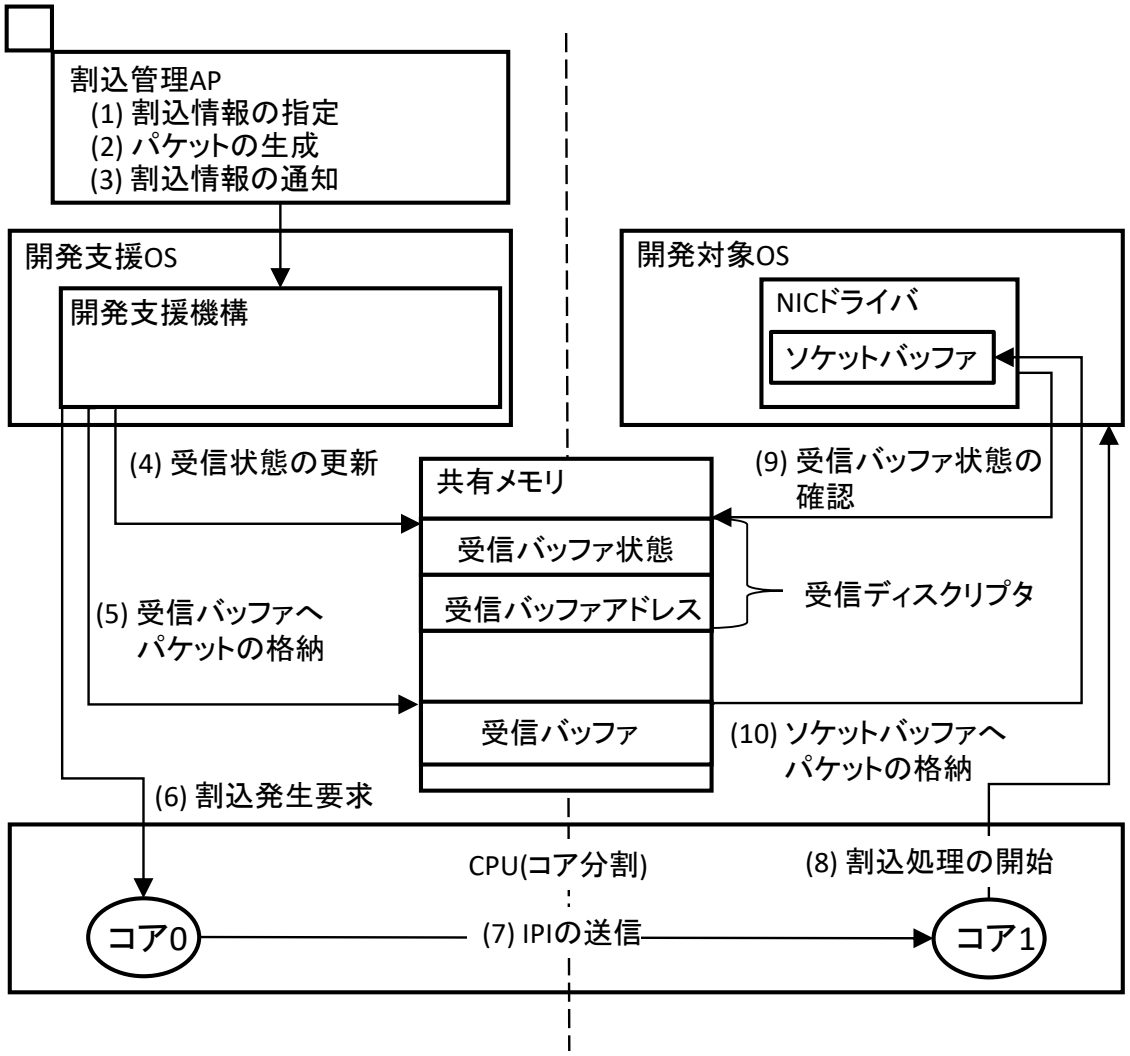

図 3.2 NIC ハードウェアを擬似対象とした開発支援環境の処理流れ

3.5 処理流れ

本開発支援環境の処理流れを図 3.2 に示し，以下で説明する．

(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を更新する．

(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 の発行要求を受けて，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) 開発対象 OS の占有するコア 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．

第 4 章

評価

4.1 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 ハードウェア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構築の影響が局所化できていることを示す．

さらに，ドライバへの適用評価として，NIC ドライバの性能を評価し，評価環境においてどの程度の通信速度に対応可能かを示す．これにより，本開発支援環境がドライバのテストに有用であることを示す．

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.

4.2 評価環境

本稿における評価環境を表 4.1 に示す.

表 4.1 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:256MB, 開発対象 OS:256MB
NIC ドライバ	RTL8169

4.3 性能評価

4.3.1 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，本開発支援環境が大きな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 4.2 に示し，以下で説明する.

(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 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

表 4.2 開発支援環境で実現できるパケットサイズの評価

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

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 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 は 16KB までのパケット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 ドライバの性能の限界を試験可能である．

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，本開発支援環境は大きな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．

4.3.2 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 3.4.3 項に示している．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 4.3 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 4.3 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

4.4 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をするのは 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．具体的な改変箇所は 3.4.4 項に示している．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 4.4 に示す．表 4.4 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 4.4 NIC ドライバの改変行数

内容	追加	削除
各種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

4.5 ドライバへの適用評価

4.5.1 通信処理の性能

本開発支援環境を用いて NIC ドライバのテストを行う．これにより，本開発支援環境がドライバのテストに有用であることを示す．以下の 2 つの測定を行うことで開発対象 OS の NIC ドライバの性能をテストする．

(1) 処理可能なパケット挿入間隔

(2) 通信速度

これらの項目を測定することにより，本評価環境における NIC ドライバの処理性能を明らかにする．この調査を行うことで，本環境が高速な NIC ハードウェアを擬似し，NIC ハードウェアを用いずに開発対象ドライバの処理性能をテストできることを示す．本環境を用いてテストを行うことで，開発対象ドライバがどの程度高速な NIC に対応できるかが分かる．

4.5.2 処理可能なパケット挿入間隔

指定した間隔でパケット挿入処理を 5000 回動作させ，NIC ドライバでパケット受信成功率を求めることを 1 サイクルとし，パケット挿入間隔を増加させながら複数サイクル行う．これにより，割込間隔と受信成功率の関係を測定する．ここで，受信成功とは NIC ドライバ内でパケットをソケットバッファに格納することとする．パケット挿入処理にはメモリ複写処理が含まれるため，パケットサイズの増大にともなって受信成功率が 100%になる間隔が

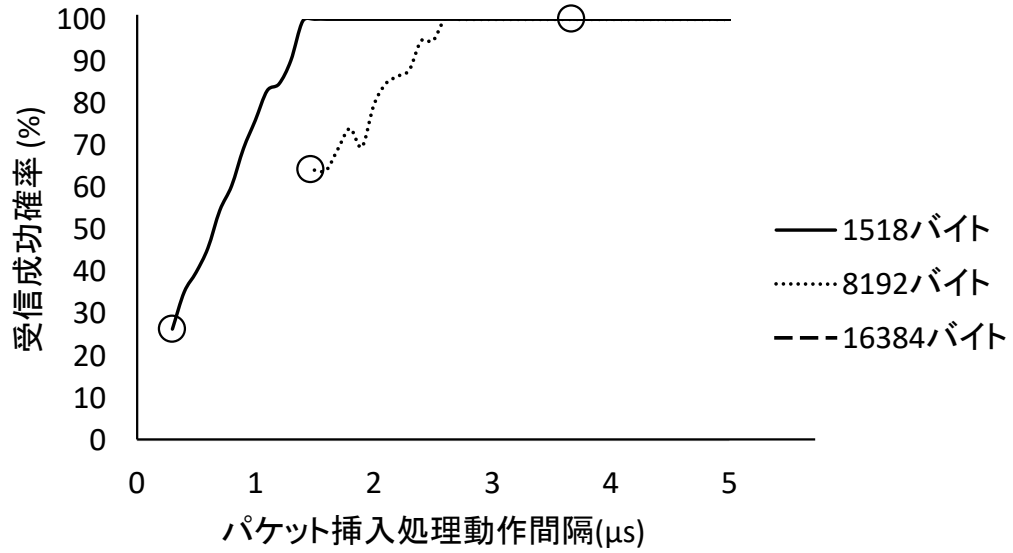

図 4.1 パケット挿入処理動作間隔と NIC ドライバにおけるパケット受信成功率の関係

長大化すると考えられる．このため，この測定を複数のパケットサイズで行う．測定に使用するパケットサイズは 4.3.2 項と同じ以下の 3 つのものを用いる．

(1) 1518 バイト (MTU のサイズ)

(2) 8192 バイト (受信バッファの半分のサイズ)

(3) 16384 バイト (受信バッファのサイズ)

結果を図 4.1 に示し，以下で説明する．なお，図中の丸印は各パケットサイズで実現可能なパケット挿入間隔の最小値 (表 4.3) を示しており，示した値以下の間隔は実現できないため，示した値以上の間隔の結果を示している．結果から，パケット挿入間隔の増加にともなって，受信成功率が 1 次関数的に増加していることが分かる．また，パケットのサイズの増加にともなって，受信成功率が 100%になるまでの間隔が長大化している．これは，パケット受信割込処理中にメモリの複写が行われるためである．

すべてのパケットを受信可能なパケット挿入間隔がパケット挿入処理の最小値に比べて大きいのは，パケット受信処理にはソケットバッファをドライバよりも上位のレイヤに送信する処理が含まれているためである．

4.5.3 通信速度

連続でパケットを挿入した際，各パケットサイズ毎に NIC ドライバで実現できる最大の通信速度を算出した．具体的には，4.5.2 項の測定において初めてパケット受信成功率が 100%となった際のパケット挿入処理 5000 回にかかった時間とパケットのサイズから各パケットにおける通信速度を算出した．

評価にあたり，NIC ハードウェアを用いて NIC ドライバで処理できる最大の通信速度を測定した．具体的には，2 台の計算機を LAN ケーブルで接続し，一方の計算機から他方の計算機へパケットを連続で挿入した際の通信速度を測定した．測定においてパケットサイズは MTU である 1518 バイトとした．

結果を表 4.5 に示し，以下で説明する．表 4.5 から提案手法を用いた際，NIC ハードウェアを大きく超える通信速度を実現できていることがわかる．また，最速で約 30Gbps を実現できており，実際の NIC ハードウェアより高速な NIC ハードウェアを擬似できていることが分かる．

本評価環境ではメモリのみを用いて通信を行なっているため，このような膨大な通信速度を実現できる．測定を行った計算機のメモリ帯域幅は約 130Gbps である．また，パケット受信処理にはメモリ複写以外に長い時間を費やす処理が含まれているため，本測定は妥当な結果であるといえる．

これらの結果から，新規 NIC ドライバ開発において開発対象ドライバを高速な NIC ハードウェアに対応させる際に本環境を用いることで，NIC ハードウェアを用いずに開発対象ドライバの処理性能を測定できると考えられる．

表 4.5 各パケットサイズにおける実現可能な通信速度

パケットサイズ (B)	通信速度 (Gbps)
(NIC) 1518	0.9
1518	6.3
8192	22.7
16384	30.6

4.6 有用性評価

4.6.1 実施するテストの検討

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
パケッ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．

4.6.2 テスト対象カーネルの準備

(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

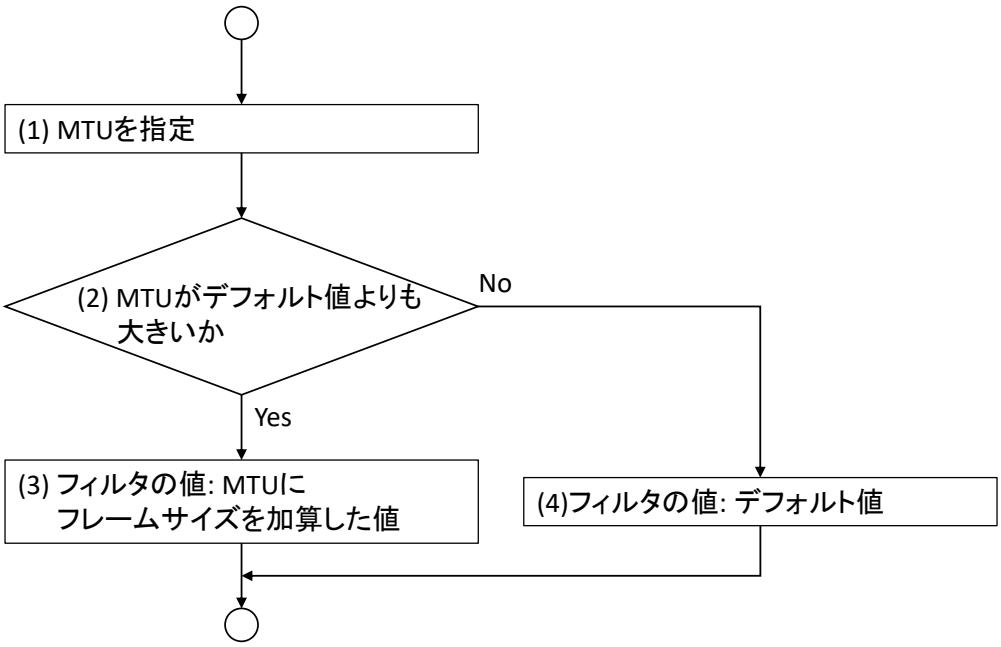

図 4.2 バグを含んだハードウェアフィルタ設定の処理流れ

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

4.6.3 特定のサイズのパケットを破棄してしまうバグ (バグ 1)

本バグは Linux 2.6.31 の NIC ドライバである 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 4.2 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．

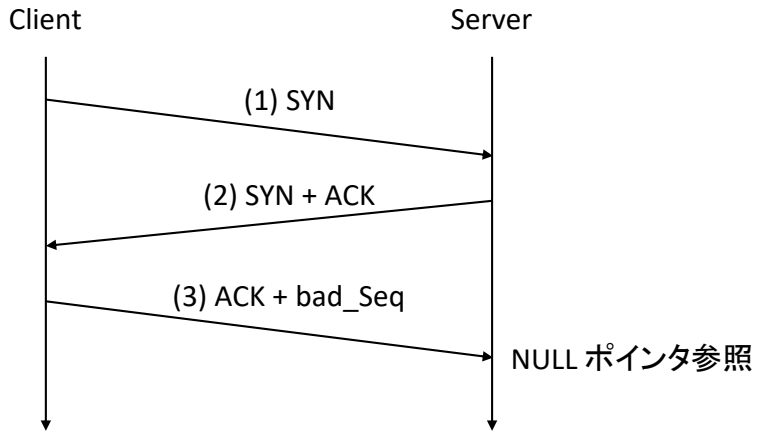

図 4.3 特定のヘッダのパケットを受信し，カーネルパニックを発生するまでの処理流れ

(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．

4.6.4 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 (バグ 2)

本バグは Linux 2.6.27-rc2 の TCP ハンドラに存在したバグである．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．本バグは文献 [6] で紹介されていたバグである．本バグが発生するまでの処理流れについて図 4.3 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 ACK パケットと SYN パケットが送信される．

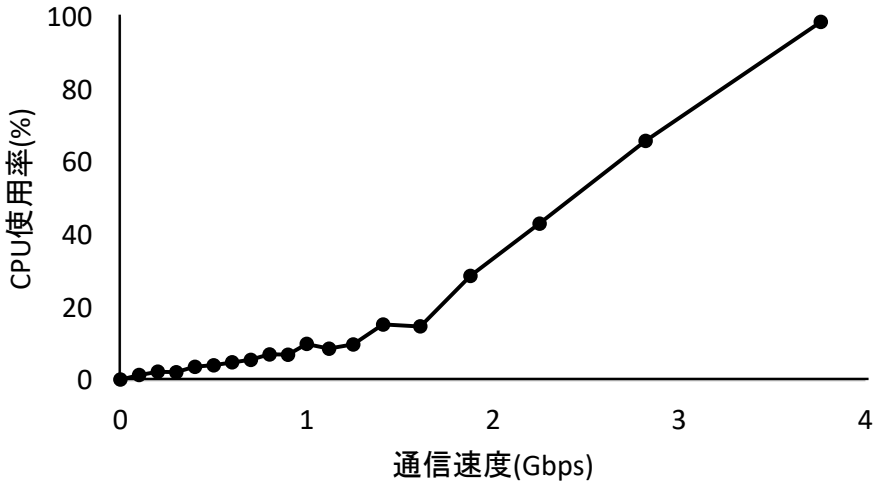

図 4.4 UDP フラッド攻撃時の CPU 使用率

(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．

4.6.5 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 4.4 に示す．結果から 0〜2Gbps 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．

4.7 考察

まず，性能評価から，本環境では実際の NIC ハードウェアを用いた場合よりもパケットに関する情報を自由に指定可能であることが分かる．本環境で用いた NIC ハードウェアよりも大きなパケットを高速に受信できるため，より高性能な NIC ハードウェアを想定した開発が可能であると考えられる．また，基本評価で示した範囲でパケットサイズと間隔を調整することで任意の性能の NIC ハードウェアを擬似することも可能である．これにより，開発したい NIC ハードウェアを用いずにドライバやプロトコルスタックの開発を支援できると考えられる．

次に，実装コスト評価において，本開発支援環境の構築における開発対象 OS の改変箇所をごくわずかに抑えられていることから，開発支援環境構築における開発対象 OS への影響を局所化できていると言える．

さらに，NIC ドライバへの適用評価から，NIC ドライバにおいて処理できる限界のパケット挿入間隔を調査できることを示した．本稿で示した環境においては最速で 30.6Gbps の速度を実現できることが分かった．これにより，本開発支援環境はドライバをテストできることを示した．

最後に，有用性評価から，本環境で指定可能にしたパケットサイズ，ヘッダとデータ，およびパケット挿入間隔を用いて OS をテストできることを示した．例として示したテストを行うことで 2 つのバグを再現することができた．これにより，パケットサイズやヘッダ，データ等を網羅的に変化させ，OS の挙動を確認するようなテストに本環境が有用であると考えられる．高速にパケットを受信させることで UDP フラッド攻撃を再現し，その影響を調査可能であることを示した．これにより，本環境が高速な 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した．

第 5 章

関連研究

5.1 デバイスドライバの開発を目的とした研究

デバイスドライバの開発を目的とした研究として SymDrive[7] がある．これはデバイスを用いずにドライバのコードを網羅的に実行することでバグを発見し，開発を支援する研究である．しかし，SymDrive では実際に割込を発生させられないため，ストレステストやパフォーマンスチューニングのようなテストには使用できない．本研究も SymDrive と同様にデバイスは用いないが実際に割込を発生させられる．これにより，短い間隔で大量のパケットを受信でき，ストレステスト等に使用できる．

5.2 OS の動作の保存/再現を目的とした研究

割込処理の開発を目的とした研究として LoRe[8] がある．LoRe は OS の動作を保存，再現することで再現の困難な割込処理を再現し，開発を支援する．保存と再現を実現するため，LoRe では仮想計算機を用いて OS の動作を監視している．LoRe の特徴としてログのサイズを従来の手法と比べて削減したこと，再現にかかる時間を削減したことがある．しかし，LoRe では仮想計算機を用いているため，仮想化による処理オーバヘッドのため，実際に発生した割込間隔よりも長い間隔となってしまう．本研究では Mint を用いることで仮想化を用いず OS を動作している．したがって，実際に発生する割込間隔より短い間隔を実現できる．

5.3 バグの再現を目的とした研究

割込処理におけるバグの再現を目的とした研究として仮想マシンモニタを用いた割込処理のデバッグ手法 [6] がある．このデバッグ手法は OS を仮想計算機上で動作させる．割込を発生させたいコード位置にハイパーコールを挿入し動作させることで任意のコード位置で割込を発生させる．これにより，再現させたいバグが発生する任意のタイミングで割込を発生させられる．しかし，このデバッグ手法で短い割込間隔に依存するようなバグを再現しようとした場合，ハイパーコールの間隔を開発者が調整することになり，非常に困難である．本研究では割込間隔を指定可能であるため，容易に任意の間隔を実現できる．

5.4 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 packetdorill[9] 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．packetdorill 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 ドライバやハードウェアインタフェース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うことが可能である．

5.5 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 [10] がある．文献 [10] 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 [10] ではソフトウェアベースのツールである Hping3[11] と文献 [10] の提案手法の速度を比較している．結果，文献 [10] 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 [10] 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 [10] 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．

5.6 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[12] 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 [13]．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を実現できない．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．

第 6 章

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することで，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等の情報を制御できる．これにより，高速で大量のパケットを受信させることができ，高速な NIC ハードウェアを想定したテストが可能になる．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 16384 バイト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，パケットが 1518 バイトの場合，最速で約 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．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．本開発支援環境のドライバへのテストの有用性を示すため，NIC ドライバの通信性能を評価した．本稿で示した評価環境であれば最速で 30.6Gbps を実現できる．さらに，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，Linux 既存のバグを 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．

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, ならびに佐藤将也助教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．最後に，本研究を進めるにあたり，経済的，精神的な支えとなった家族に感謝いたします.

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] 松本直人：x86 サーバにおける 40Gigabit Ethernet 性能測定と課題，情報処理学会研究報告， Vol. 2013-IOT-22, No. 1, pp. 1–3 (2013).

[6] 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).

[7] Renzelmann, M. J., Kadav, A. and Swift, M. M.: SymDrive: Testing Drivers without Devices., *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pp. 4–6 (2012).

[8] Li, J., Si, S., Li, B., Cui, L. and Zheng, J.: LoRe: Supporting Non-deterministic Events Logging and Replay for KVM Virtual Machines, *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous*

Computing (HPCC_EUC), 2013 IEEE 10th International Conference on, pp. 442–449 (2013).

[9] 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).

[10] 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).

[11] Sanfilippo, S.: Hping3, (online), available from ⟨http://www.hping.org/hping3.html⟩ (accessed 2017-01-08).

[12] : SR-IOV, PCI Special Interest Group (online), available from ⟨http://pcisig.com/⟩ (accessed 2017-01-25).

[13] 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).

発表論文

[1] 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，第 136 回システムソフトウェアとオペレーティング・システム研究会，pp. 1-8 (2016).

[2] 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通信処理の開発支援環境の実現と評価，情報処理学会研究報告，Vol. 2017-OS-139, No. 3，第 139 回システムソフトウェアとオペレーティング・システム研究会，pp. 1-8 (2017，掲載予定).