不正侵入検知システムにおけるマルチコア上でのシグネチャ割当によるレイテンシ削減手法

山田正平[†1]　見神広紀[†1]　木村啓二[†1]　笠原博徳[†1]

概要：企業や政府機関を標的としたサイバー攻撃が年々高度で大規模なものになっている．これらサイバー攻撃の有効策のひとつとして不正侵入検知システムが挙げられる．不正侵入検知システムはネットワークを監視し，IP パケットをフィルタリングすることで不審なアクセスをリアルタイムで検知する．一方で，膨大なパケットを処理するための処理性能が求められる．そこで本研究では，シグネチャ型の不正侵入検知システムにおいてシグネチャを分割し，マルチコアへの割当によるレイテンシ削減手法を提案する．本手法は，並列処理によってパケットあたりの検知処理時間の短縮が可能である．レイテンシ削減手法をオープンソースの不正侵入検知システムである Suricata において適用し，DARPA Intrusion Detection Evaluation Data Set などのデータセットを入力とした際の検知処理性能を評価した．その結果，2 コア上でシグネチャを分割しない場合と比較して DARPA Intrusion Detection Evaluation Data Set において 4 コア上で最大 3.22 倍の検知処理時間の短縮を得ることができた．

A Latency Reduction Technique for IDS by Allocating Decomposed Signature on Multi-core

SHOHEI YAMADA[†1]　HIROKI MIKAMI[†1]
KEIJI KIMURA[†1]　HIRONORI KASAHARA[†1]

Abstract: Cyber attacks targeting on companies and government organizations have been increasing and highly developed. An Intrusion Detection System (IDS) is one of efficient solutions to prevent those attacks. An IDS detects illegal network accesses in realtime by monitoring the network and filtering suspicious IP packets. Large processing performance is required for IDSs to process a large number of IP packets in realtime. In order to satisfy this requirement, a latency reduction technique for signature-based IDSs by allocating decomposed signature on multicores is proposed in this paper. The proposed technique is implemented in Suricata, which is an open source IDS, and evaluated it with several data sets, such as DARPA Intrusion Detection Evaluation Data Set. The evaluation results show the proposed techniques with four cores achieves 3.22 times performance improvement in maximum comparing with two cores without signature decomposition.

1. はじめに

情報化社会の発展により，インターネットを介して様々な情報をやりとりすることが当たり前の社会となった．一方で秘密情報を悪用,収集しようとする者も存在しており，この者たちは脆弱性と呼ばれるプログラムの欠陥を攻撃することで強引に秘密情報へアクセスを試みる．これはサイバー攻撃と呼ばれ，近年では企業や政府機関を対象として技術情報や国家機密情報が漏洩する事件も発生している．これらサイバー攻撃の有効策のひとつとして不正侵入検知システムが挙げられる．不正侵入検知システムは外部や内部からのアクセスを問わず，ネットワークに流れる情報の断片である IP パケットを監視する．このとき，IP パケットの中身をシグネチャと呼ばれる攻撃や不正アクセスなどに見られるパターンを定義したものと比較・判定することで，脆弱性を攻撃するアクセスやマルウェアによる不正アクセスなどを検知し，管理者へ通報する．

一方で,不正侵入検知システムは高速通信網の普及に伴い，処理性能の向上が求められる．この処理性能の要求に対し，マルチコア上の並列処理や複数台のマシンを用いた分散処理手法が提案されている[1]．並列処理においては近年普及している 1 つのチップに複数のプロセッサコアを搭載したマルチコアプロセッサ，及びマルチコアプロセッサを複数用いたマルチプロセッサアーキテクチャが用いられる．マルチコアプロセッサは集積度の向上により比較的安価で低消費電力であり，複数のプロセッサコアに処理を並列で処理させることで 1 台のマシンのみでも処理性能の向上が期待できる．

本稿では,マルチコアプロセッサを用いた並列処理によるシグネチャ型不正侵入検知システムのレイテンシ削減による高速化を検討・提案する．

†1 早稲田大学
Waseda University

2. 不正侵入検知システムの概要

本章ではネットワークセキュリティシステムである不正侵入検知システム（IDS: Intrusion Detection System）の概要と本報告の評価対象である Suricata について述べる．

2.1 不正侵入検知システムの原理

IDS はネットワークを流れるパケットを収集し，シグネチャ内に定義された文字列とパケット内のバイナリ文字列に対してパターンマッチングによる比較処理を行う．定義された照合パターンと一致したパケットから不正アクセスを検知し，ログの記録とネットワーク管理者への通報を行う．また，このようなシグネチャを用いる IDS をシグネチャ型と呼ぶ．一般的に，シグネチャ数が増えるほどパターンマッチングには時間がかかる．一方で，対象のネットワークを流れるパケットをリアルタイムで処理できる高速処理能力が求められる．図 1 に IDS の概要図を示す．

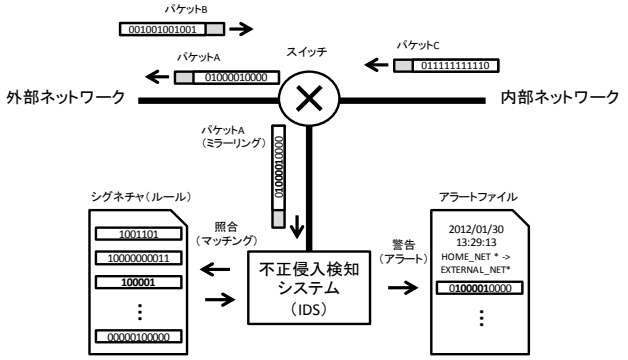

図 1　不正侵入検知システムの概要図

図 1 において，スイッチのミラーリング機能によって監視ネットワークを流れるパケットが IDS へ入力される．この入力パケットに対して，IDS はインクルードされたシグネチャとマッチング処理を行う．この結果，不審なパケットと判定された場合に管理者へ通報し，その内容をファイルへ記録する．

2.2　不正侵入検知システム Suricata の概要

Suricata は 2010 年に登場した，C 言語で記述されたマルチスレッド対応のシグネチャ型 IDS である[1]．オープンソースであり，米国安全保障省が支援する Open Information Security Foundaiton （OISF）により開発される[2]．なお，シグネチャは別のオープンソースコミュニティである Emerging Threat （ET）より提供される[3]．Suricata はマルチスレッドによってパケットの処理をパイプライン方式で行う．

パイプラインを構成する各ステージの役割は次のとおりである．

Decode： Decode における処理はネットワークを流れるパケットをキャプチャし，Suricata が処理する形式に変換することである．図 1 のようにスイッチからのミラーリングパケットが IDS で受信される時に Decode 処理がパケットをキャプチャする．

Stream： Stream における処理は Decode 処理によりキャプチャされたパケットをシーケンス番号に基づき再構築する．これは不正侵入パターンをパケットの細分化によって検知回避する TCP フラグメンテーション攻撃を防ぐためである．

Detect： Detect における処理はキャプチャされたパケットに対してシグネチャとの比較と検知判定を行う．パケットデータをバイナリの文字列と見なし，シグネチャ内で定義された照合パターンとパターンマッチング処理を行うことで不正侵入を検知する．Aho-Corasick 法をはじめとしたパターンマッチングアルゴリズムが実装され，これらはステートマシンを用いて一度に効率よくパターンマッチングを行うことが可能である．しかしながら，ステートマシンの特性上，アルゴリズム自体の並列化は困難である．

また，一般的に IDS はこのパターンマッチングを含めた検知処理時間が処理の大部分を占め，処理のボトルネックとなっている．

Output： Output における処理は Detect 処理の結果をもとにアラートとログの出力を行う．

図 2 に Suricata のプログラムアーキテクチャである PacketPipeline アーキテクチャを示す[4]．図 2 の Scheduler と Detect 処理の関係については 2.3 節で詳しく述べる．

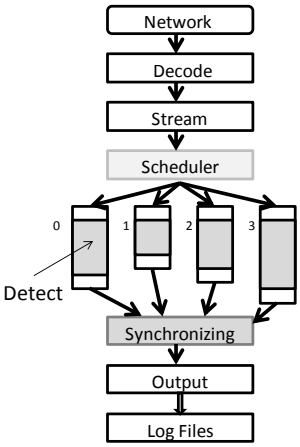

図 2　PacketPipeline アーキテクチャ

2.3　不正侵入検知システムの性能評価指標

ここで IDS の性能評価指標，及び IDS 高速化に伴うトレードオフについて述べる．

レイテンシ： レイテンシとは入力から出力までの 1 サイクルの処理時間を指す．Suricata の場合にはパケットデータが Decode，Stream，Detect，Output を経た一連の処理時間を指す．仮にレイテンシが半分になれば，パケットあたりの処理時間が半分になるということになり，全体の処理速度は 2 倍に向上する．

スループット： スループットとは単位時間における処理性能を指す．単位時間あたりの処理性能を上げる方法のひとつに同一の処理を別のデータに対して複数並列に行うことが挙げられる[5]．この方法は IDS においても広く利用され，Suricata は Detect スレッドを複数用いることでスループット向上を図る実装がされている[6]．

Suricata では，図 3 のように Stream 処理が再構築した複数のパケット p1，p2，p3 が decode-queue にある場合，スケジューラは複数の Detect スレッドへパケットを割当てることで Suricata はスループット性能を確保している．一方で，パケットあたりの検知処理時間は Detect スレッド数に関わらず変化しないことが分かる．

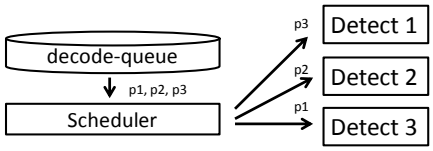

図 3　Suricata 上スケジューラ処理

処理性能の向上を図る上で次のようなトレードが存在する．即ち，レイテンシ削減は処理が可能なパケットが 1 つからでも性能向上が可能であるが，IDS のような元々低レイテンシである処理では性能向上が難しい．そのため，過去の取り組みが少ない．対してスループット向上は高い性能向上を期待できる．一方で，トラフィック量が少ない状況や IDS のパケット再構築処理の結果よっては並列処理可能なパケットデータの数が限られる．このとき，IDS の負荷が小さいにもかかわらずリソースをいくら増やしても性能は向上しない．

本研究ではレイテンシ削減に焦点を当て，高速化手法の検討と提案を行う．

3. レイテンシ削減による IDS 高速化手法

本章では，研究対象である Suricata におけるレイテンシ削減手法の提案する．

一般的に，シグネチャ型 IDS では検知処理時間が処理の大部分を占める．これは数千やそれ以上のシグネチャをパケットに対して適用するためである．

提案手法ではプログラムアーキテクチャの観点からシグネチャを分割し，これら分割したシグネチャをマルチコア上に割当てることにより，検知処理のレイテンシを削減する．

3.1 シグネチャの分割

まず，ルールセットとして配布されるシグネチャの分割について検討する．なお，シグネチャの分割を検討する上でシグネチャの数やシグネチャごとのアラート数は処理時間に必ずしも影響しない．これは照合パターンごとのパターンマッチングの処理時間の違いに加え，トラフィックによって照合される頻度が異なるためである．このことから，パターンマッチング時間が大きいシグネチャでも照合される頻度が少なければ処理時間への影響は小さい．逆にパターンマッチング時間が小さいシグネチャでも照合される頻度が多ければ，処理時間への影響は大きい．以上から，シグネチャの影響度はトラフィックの内容に依存してしまう．

ここでシグネチャを分割する上でプログラム内の検知処理フローに着目する．検知処理フローは主にパターンマッチングとイベントチェックに分かれており，Suricata はこれらを逐次的に処理している．

3.1.1 パターンマッチング

パターンマッチングを行う上で Suricata は冗長なパターンマッチングを防ぐため，図 4 の関数 DetectMpmPrefilter に示すような複数回のパターンマッチングを逐次的に行うプログラム構造をもつ．

```
static inline void DetectMpmPrefilter ( ){
    if( ){      // ペイロード対象
        PacketPatternSearch( );
    }
    …
    if( ) {       // HTTP ヘッダ・メッセージ対象
        …
        if ( ){
            DetectEngineRunHttpHeaderMpm( );
        }
        if ( ){
            DetectEngineRunHttpRawHeaderMpm( );
        }

        …

        If( ){
            DetectEngineRunHttpCookieMpm( );
        }
    }
}
```

図 4　DetectMpmPrefilter のプログラム構造

図 4 における DetectMpmPrefilter 内の PacketPatternSearch などの関数群は if 条件を満たして実行された場合にパターンマッチングを実行する．

一方で，シグネチャにはパケットデータ全体のどのデータを対象とした照合パターンであるかが定義されており，このパターンマッチングの対象別に関数が実装されている．これにより，インクルードされたシグネチャとは無関係なデータに対してはパターンマッチングを行わないが，回数としてはパケットごとに複数回のパターンマッチングを実行することになる．ここでシグネチャ定義を考慮すると，ペイロードを対象にしたシグネチャ，HTTP ヘッダ・メッセージを対象にしたシグネチャの 2 つに分類することが可能である．図 5 に実際の Suricata のシグネチャ例を示す．

```
alert http $HOME_NET any ->
$EXTERNAL_NET any (msg:"ET POLICY
GridinSoft.com Software Version Check";
flow:established,to_server;
content:"User-Agent|3A 20|GridinSoft";
http_header; classtype:trojan-activity;
sid:2013719; rev:3;)
```

図 5　Suricata のシグネチャ例

シグネチャの照合パターンは content によって定義される．この content による照合パターンの定義は標準ではペイロードを対象とする．図 5 では content で定義された照合パターンに対して http_header オプションが付与されている．このオプションにより，図 5 のシグネチャの場合にはペイロードではなく HTTP ヘッダ・メッセージ部分を対象とした照合パターンが定義されている．

以上から，これら content に付与されるオプションによってシグネチャがペイロードを対象にしたものか，HTTP ヘッダ・メッセージを対象にしたものかで分類可能である．このシグネチャの分類を適用することによって，検知処理における複数回のパターンマッチングを並列で実行可能である．

3.1.2 イベントチェック

イベントチェックではパターンマッチングを必要としないシグネチャの判定を行う．パターンマッチングを必要としないのはパケットのヘッダフィールドやパラメータ情報などから不審なパケットを判定するためである．また，プロトコルを問わずマッチングされるシグネチャも含まれる．

これらのシグネチャをイベントチェックとして分類し，図 6 にイベントチェック時に適用されるシグネチャの例を示す．図 6 のシグネチャはパケットのプロトコルや方向と問わず，IPv4 パケットのパケットサイズが規定よりも大きい際にアラートを生成する．

```
alert pkthdr any any -> any any
(msg:"SURICATA FRAG IPv4 Packet size too
large"; decode-event:ipv4.frag_too_large;
sid:2200069; rev:1;)
```

図 6 イベントチェックを対象としたシグネチャ例

また，イベントチェクに分類されるシグネチャは定義内容からいくつか種類がある．表 1 にこれらイベントチェックの種類を示す．このイベントチェックにおいても種類ごとにシグネチャを分割することで並列実行可能である．

表 1　イベントチェックの種類

種類	対象
decode-event	プロトコルヘッダのデコード内容
stream-event	TCPプロトコルのシーケンス内容
app-layer-event	アプリケーション層プロトコル内容
*-csum	IPv4, IPv6パケットのチェックサム
threshold	パラメータに対するしきい値チェック
byte_test	バイトフィールドのテスト
flags	パケットヘッダに対するフラグチェック

以上から，ペイロード，HTTP ヘッダ・メッセージ，イベントチェックの分類ごとにシグネチャを分割，マルチコア上で並列処理させることで検知処理時間のレイテンシ削減を実現する．

3.2　マルチコア上へのシグネチャ割当

次に，3 つに分類したシグネチャをマルチコアに割当てることを考える．オリジナルの Suricata ではシグネチャを分割してインクルードすることはできない．よって，目的の機能を実現するための実装を行った．図 9 に実装したマルチパイプライン版 Suricata のアーキテクチャを示す．

このマルチパイプライン版 Suricata は図 2 に示した PacketPipeline アーキテクチャを fork 関数によって複製したものである．各々の PacketPipeline の Detect スレッドは 1 つであり，事前に分割したシグネチャがインクルードされる．また，PacketPipeline 同士は共通のパケットを非同期で処理することで，同期のオーバーヘッドは発生しない．

また，図 7 における各スレッドのプロセッサコアに対する割当を図 8 に示す．

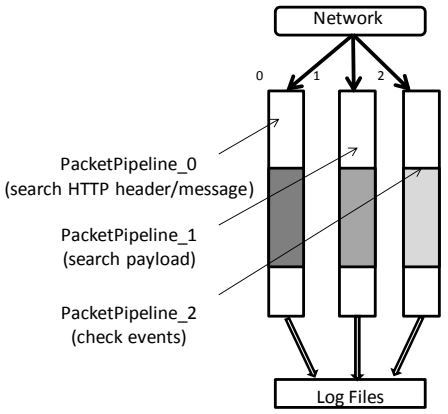

図 7　マルチパイプライン版 Suricata（3 並列実行時）

コア0: 管理スレッド（PacketPipeline_0-2）
コア1: Detectスレッド（PacketPipeline_0）
コア2: Detectスレッド（PacketPipeline_1）
コア3: Detectスレッド（PacketPipeline_2）

図 8　マルチパイプライン版 Suricata におけるコアの割当

マルチパイプライン版 Suricata はシグネチャの分割数に応じて PacketPipeline を最大 3 つ複製し，PacketPipeline の数だけ使用コア数が増加する．PacketPipeline 自体はマルチスレッドであるため，最も負荷の高くなる Detect スレッドはそれぞれ 1 つのコアへ，管理スレッドと呼ばれる Detect スレッド以外のスレッドは全てコア 0 へ割当てる．

また，2 コアで動作をする場合にはシグネチャは分割されず，スレッド数もオリジナル版の Suricata と共通である．

3.3　シグネチャ割当の最適化

シグネチャ分類の特性上，マルチパイプライン版 Suricata は監視するトラフィック内容によってはパイプラインの処理量に大きな差が生じる．これを軽減するために監視するトラフィックの傾向に応じたシグネチャ割当の最適化とその手順を示す．また，最適化の際には監視トラフィックのパケットキャプチャデータを使用する．

step 1： 監視トラフィック上でペイロード，HTTP ヘッダ・メッセージ，各イベントチェックの種類において，どのくらいの処理差があるかを調査する．これは Suricata の -r オプションによって提供されるパケットキャプチャデータ読込機能を使用する．

この機能は IDS 導入の際に，事前に取得したトラフィックデータを読み込ませることで IDS の処理能力やアラートの確認などのために提供される．

この-r オプションを用いてペイロード，HTTP ヘッダ・メッセージ，各イベントチェックの種類ごとに処理時間を計測し，検知処理時間を算出する．この検知処理時間をシグネチャの分類と種類ごとのコストとする．

なお，検知処理時間の算出方法は 4 章の性能評価のものと同一である．

step 2： step1 で得られたコストを用いて各 PE への割当を行う．コアへの割当については並列処理における一般的なスケジューリング手法であるクリティカルパス法と同様の手順で割当を行う[7]．なお，シグネチャ間に依存はないため依存に関しては考慮しなくてよい．

また，クリティカルパス法は一様なコスト分布と十分な要素数があれば均等なコストを PE ごとに割当てることが可能である．一方で，今回はシグネチャの分類と種類を全て合わせても要素数は 10 程度であるため，コストのばらつきによってはパイプラインの処理量に差が生じることはやむを得ない．

割当アルゴリズムは次のとおりである．

1. 要素をコスト順に降順ソート
2. 要素が割当てられるコア番号の決定
   ・このとき，コア番号に対応したコストの合計値が最小であるコア番号を選択
3. 要素のコストを加算後、次の要素を参照
4. 要素がなくなるまで手順 2 と 3 を繰り返す

4.　性能評価

本章では，様々なデータセットを対象にシグネチャ分割を行ったマルチパイプライン版 Suricata を Intel Xeon プロセッサ上で評価した結果について述べる．本評価の目的は提案するマルチコア上でのシグネチャ割当によるレイテンシ削減手法による検知処理速度向上の有効性の確認である．

4.1　評価環境と評価対象

本評価では表 2 に示す計算機環境を使用した．

表 2　評価環境

OS	Ubuntu 10.04 LTS (64bit, Linux 2.6.32)
CPU	Intel Xeon L5630 (2.13GHz, 4Core) * 2Chips
PE	8PE
L2 cache	1MB /Chip
L3 cache	12MB / Chip
Compiler	GCC-4.4.6
RAM	24GB

さらに, 評価対象である IDS とシグネチャを表 3 に示す.

表 3　評価対象

IDS	Suricata-1.4.5(original)
	Suricata-1.4.5(multi-pipeline)
Option	-c -r
Rule set	emerging-rules (2013/09/01)
Signatures	13,799 signatures

また，Suricata はバッファサイズなどの様々なパラメータを詳細設定することが可能であるが，詳細設定に関してはオリジナル版とマルチパイプライン版でデフォルトのものを使用した．ただし，スケジューラは PacketPipeline あたりの Detect スレッド数が 1 の場合には single モード，それ以外の場合には autofp モードを使用する．

4.2　評価方法

本評価はネットワーク環境を使用せず，ローカルディスク上の評価用データを Suricata の読込機能を使用することによって性能を評価する．

ネットワーク環境ではネットワーク機器や評価環境のネットワークインタフェースの性能によっては IDS に十分な負荷がかけられない可能性がある．一方で，ファイル読込機能は IDS の処理能力が許す限りパケットを入力できるが，ファイルの読込レイテンシはネットワークからパケットが入力されるときよりも大きい．また，処理終了時に出力される処理時間にはこのファイル読込時間も含まれるため，本評価では検知処理能力を評価する上で次式によって検知処理時間を算出した．

$$\text{検知処理時間} t_D = \text{出力時間} t - \text{読込時間} t_R$$

この中で，読込時間 t_R は事前に取得したインクルードするシグネチャ数を 0 個とした場合の Suricata の出力時間である．これは，インクルードするシグネチャ数が 0 であるため，Suricata は検知処理を実行せずにファイルの読込だけを行うためである．

また，アラート出力処理の時間を考慮していないが，アラートの出力は別スレッドがネットワーク，ファイル読込問わず低レイテンシで行っているため, 今回は考慮しない．

最後に，マルチパイプライン版 Suricata においては複数ある PacketPipeline で一番処理時間が大きい PacketPipeline の出力時間から検知処理時間を算出するものとする．

4.3　評価用データ

本評価は入力として研究機関が配布するデータセットや実用されるネットワーク環境のトラフィックデータを使用した．以下にデータセットの概要を示す．

DARPA Intrusion Detection Evaluation Data Set

DARPA Intrusion Detection Evaluation Data Set は米国マサチューセッツ工科大学にある Lincoln Laboratory より配布される IDS 評価用のデータセットである[8] [9]．これは 10 年以上前の実際のキャプチャデータになるが，様々な攻撃と発生時刻が解析されているため IDS 評価において現在も利用される．

このデータセットのうち， 1999 年の 5th week Monday から Friday を性能評価に用いる．また，各曜日で inside と ouside のデータに分かれているが，これらを評価用に連結し，Monday.pcap から Friday.pcap として使用する．

ITOC CDX Data Set

CDX Data Set は Information Technology and Operations Center (ITOC) より配布される 2009 年に行われた米国陸軍士官学校におけるサイバー防御演習で用いられたデータセットである[10] [11].

トラフィックの傾向としてはマルウェアによる攻撃が含まれないため, 大部分が TCP プロトコルパケットに偏っている．これはマルチパイプライン版 Suricata が苦手とするトラフィックの傾向である．また，演習用データということで同時セッション数が少ない．このため，オリジナル版 Suricata においても単位時間あたりに並列処理可能なパケット数が少なくなる．これはオリジナル版が苦手とするトラフィックの傾向となる．

以上から評価対象である IDS のいずれも苦手とするトラフィックデータの入力を想定している．

評価用データは 2009-04-21-04-06-19.dump10 から.dump19 までの配布データを評価用に連結し，dump10_dump19.pcap として使用する．

研究室のキャプチャデータ

上記の DARPA のデータセットや ITOC のデータセットは 10 年前の古いデータであったり, マルウェアの攻撃を含まないなど，現在実用されるネットワーク環境を想定していない．このことから，実用される環境として当研究室のトラフィックデータを評価用データとして使用する．

これは 2014 年 1 月のある週の月曜日から金曜日までの連続したトラフィックデータである．これを 5 分割し，それぞれ lab01.pcap から lab05.pcap として使用する．

これら各評価用データのパケット数とファイル容量の内訳を表 4 に示す.

表 4　評価用データ

ファイル名	パケット数	容量
Monday.pcap	3,667,917	783MB
Tuesday.pcap	5,962,053	899MB
Wednesday.pcap	3,473,044	800MB
Thursday.pcap	5,509,639	1.35GB
Friday.pcap	6,045,505	1.92GB
dump10_19.pcap	11,859,602	9.31GB
lab01.pcap	1,669,404	999MB
lab02.pcap	5,615,310	2.92GB
lab03.pcap	3,403,158	1.95GB
lab04.pcap	5,577,077	2.92GB
lab05.pcap	5,270,379	2.92GB

また，3.5 節で述べたシグネチャ割当の最適化は表 5 に示したファイルのみを使用して最適化を行った．

表 5　最適化用データ

データセット	最適化用データ
DARPA	Monday.pcap
ITOC	2009-04-21-04-06-19.dump10
研究室データ	lab01.pcap

最適化の上で，DARPA のデータセットに関しては HTTP ヘッダ・メッセージの検知処理負荷，ITOC のデータセットに関してはペイロードの検知処理負荷，研究室データに関してはイベントチェックの検知処理負荷がそれぞれ高かった．これらを考慮し，割当アルゴリズムを用いてシグネチャを各プロセッサコアに割当てている．

4.4　評価結果と考察

本節では，レイテンシ削減手法におけるシグネチャの分割数に応じたマルチパイプライン版 Suricata の検知処理性能、ならびに同一コア数を使用した場合のオリジナル版 Suricata との性能比較を評価用データごとに行う．

DARPA Intrusion Detection Evaluation Data Set

DARPA データセットにおいて 2 コア上でシグネチャを分割しない場合と 3 コア上でシグネチャを 2 分割，4 コア上でシグネチャを 3 分割した場合を比較する．2 コア実行時の検知処理時間に対する速度向上率を図 9 に示す．

図 9 の結果から，Monday.pcap において 4 コア上で 3 並列処理した場合には 2 コア実行時と比較して 3.22 倍の検知処理速度の向上を得ることができた．

一方で，3 並列処理において 3 倍以上の速度向上が得られた理由を考える．これは Suricata がパイプライン方式でパケットを処理しているためで，Detect スレッドの負荷によっては PacketPipeline がストールしてしまう．2 コア実行時にパイプラインストールによって余計に処理速度が遅くなったため，並列度以上に速度が向上したと考えられる．

また，Monday.pcap から Friday.pcap のいずれの入力に対してもシグネチャの分割数の増加に伴い，平均で 3 コア使用時 1.82 倍，4 コア使用時 2.69 倍の検知処理速度向上が得られた．

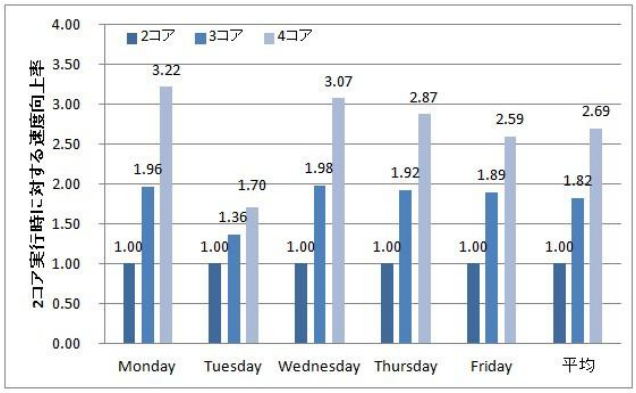

図 9　2 コア実行に対する速度向上率（DARPA）

次に，同一コア数を使用するオリジナル版 Suricata との性能を比較する．図 10 に同コア数，同並列度におけるオリジナル版 Suricata に対するマルチパイプライン版 Suricata 性能評価結果を示す．

図 10 の結果から，オリジナル版はコア数の増加に伴いマルチパイプライン版よりも速度向上率が大きくなるものの，Monday.pcap から Friday.pcap いずれの IDS 評価用データとコア数に対してもレイテンシ削減手法を適用したマルチパイプライン版が同一コア数を使用するオリジナル版に対し検知処理速度が勝る結果となった．このとき，平均で 3 コア使用時 1.49 倍，4 コア使用時 1.25 倍の検知処理速度向上が得られた．

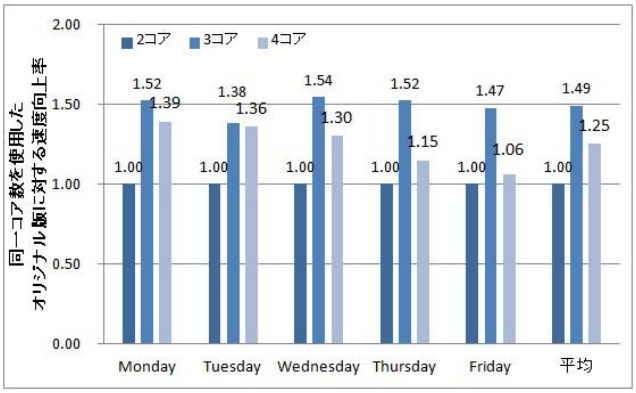

図 10　同一コア数を使用したオリジナル版 Suricata に対する速度向上率（DARPA）

ITOC CDX Data Set

ITOC のデータセットにおいて同様に 2 コア上でシグネチャを分割しない場合と 3 コア上でシグネチャを 2 分割，4 コア上でシグネチャを 3 分割した場合を比較する．この結果を図 11 に示す．

図 11 の結果から，DARPA のデータセットと比較しても速度向上率が低いことが分かる．これはトラフィックの傾向からペイロードの検知処理を行うパイプラインの処理負荷が他のパイプラインに比べ極めて大きいためである．また，HTTP ヘッダ・メッセージを含むパケットがトラフィックに含まれないため，検知処理はペイロードとイベントチェックの 2 並列処理となってしまう．このため，4 コア上でシグネチャを 3 分割したとしても検知処理速度は向上せず，最大で 1.48 倍で頭打ちとなってしまう．このような検知処理負荷が偏るトラフィックが本手法の苦手とするトラフィックである．

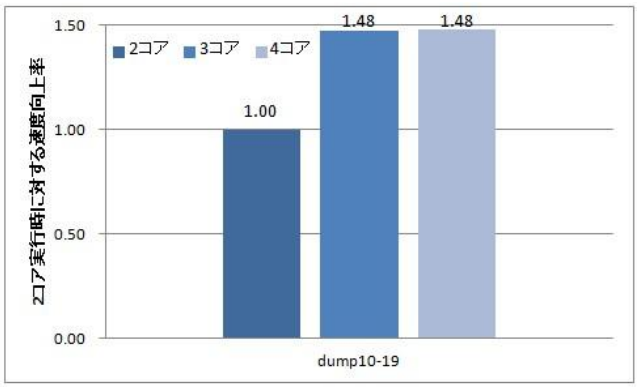

図 11　2 コア実行に対する速度向上率（ITOC）

また，同様に同一コア数を使用するオリジナル版 Suricata との性能を比較する．この性能評価結果を図 12 に示す．

一方で，コア数の増加に伴いオリジナル版は若干処理速度が向上するものの，オリジナル版も並列処理可能なパケットが少ない苦手とするトラフィックであるため速度向上率は低い．この結果，いずれのコア数に対しても図 12 のようにレイテンシ削減手法を適用したマルチパイプライン版

がオリジナル版に対し，3 コア使用時に 1.13 倍，4 コア使用時に 1.10 倍の検知処理速度が得られた．

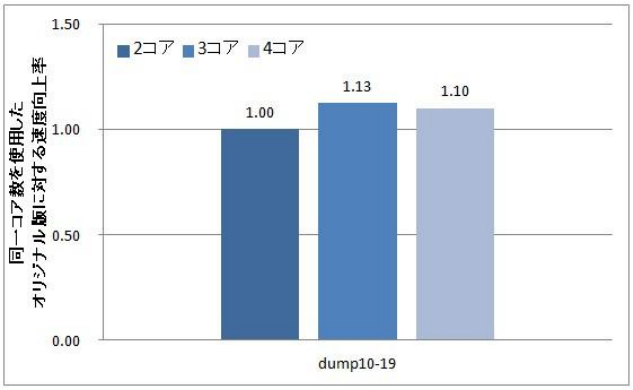

図 12　同一コア数を使用したオリジナル版 Suricata に対する速度向上率（ITOC）

研究室のキャプチャデータ

研究室のキャプチャデータに対して ITOC のデータセットにおいて同様に 2 コア上でシグネチャを分割しない場合と 3 コア上でシグネチャを 2 分割，4 コア上でシグネチャを 3 分割した場合を比較する．この結果を図 13 に示す．

図 13 の結果から，lab05.pcap において 4 コア上で 3 並列処理した場合に最大で 2.70 倍の検知処理速度の向上を得ることができた．また，lab01.pcap から lab05.pcap のいずれの入力に対してもシグネチャの分割数の増加に伴い，検知処理速度が向上した．

一方で，トラフィックの傾向としては暗号化プロトコルパケットが大きな割合を占めているため，本手法の苦手とするトラフィックであると考えられる．しかし，ITOC のデータセットに対して処理負荷の高いパイプラインは HTTP ヘッダ・メッセージやペイロードではなくイベントチェックであるため，さらにシグネチャを分割することで負荷を分散させることが可能であった．このことから，検知処理速度が頭打ちにはならず，シグネチャの分割の増加に伴い，平均で 3 コア使用時に 1.82 倍，4 コア使用時に 2.63 倍の検知処理速度向上が得られた．

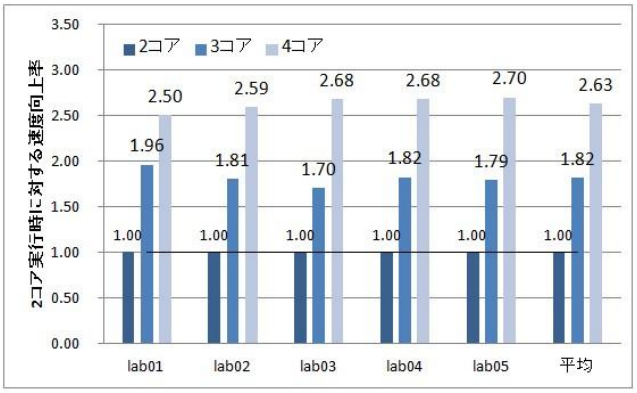

図 13　2 コア実行に対する速度向上率（研究室データ）

また，同様に同一コア数を使用するオリジナル版 Suricata との性能を比較する．この性能評価結果を図 16 に示す．これもまた同様に，実用されるネットワークデータにおいてもいずれのコア数においてもレイテンシ削減手法を適用したマルチパイプライン版がオリジナル版に対し検知処理速度が勝る結果となった．このとき，平均で 3 コア使用時に 1.49 倍，4 コア使用時に 1.25 倍の検知処理速度向上が得られた．

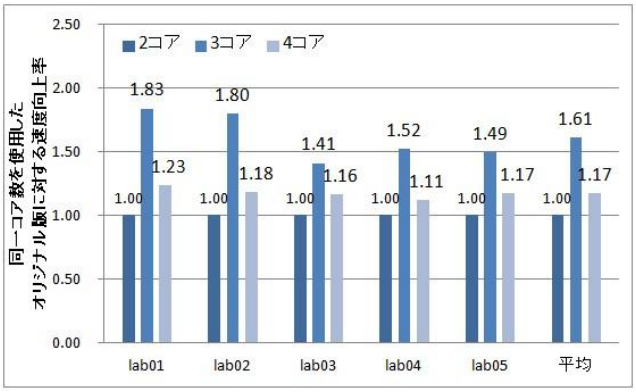

図 13　同一コア数を使用したオリジナル版 Suricata に対する速度向上率（研究室データ）

5. まとめ

本稿では，マルチコアプロセッサを用いた並列処理によるシグネチャ型 IDS の高速化において，シグネチャ分割によるレイテンシ削減手法，およびトラフィックの傾向からマルチコア上へのシグネチャ割当の最適化手法を提案した．本手法をオープンソースのシグネチャ型 IDS である Suricata に適用し，適用のために実装したマルチパイプライン版 Suricata の性能評価を行った．この結果，DARPA Intrusion Detection Evaluation Data Set や ITOC CDX Data Set，研究室ネットワークのトラフィックデータにおいて，シグネチャ分割によって 4 コア使用時に 2 コア実行時と比較して，最大 3.22 倍の検知処理時間の短縮を確認した．また，同一コア数を使用するオリジナル版の Suricata に対し，いずれのトラフィックデータ・コア数においても検知処理時間はマルチパイプライン版 Suricata が高速であった．このとき，研究室のトラフィックデータに対し，3 コア実行時のオリジナル版に比べ最大 1.83 倍の速度向上を得ることができた．以上から，従来の並列処理では性能向上が難しいトラフィックを含む，多くのトラフィックに対して本手法によるレイテンシ削減が有効であることが分かった．今後の課題としてはレイテンシ削減では処理しきれないトラフィックに対し，動的にスループット優先処理に切替えることで常に最大限の処理能力を実現する，ベストエフォート型 IDS の構築が課題となる．

謝辞　早稲田大学情報理工学専攻である後藤滋樹教授，森達也准教授から本研究に関して貴重な御助言をいただき誠に感謝致します．

参考文献

1)　大山昇吾，藤野毅: 組み込みマイコンを用いた分散型侵入検知システム，電子情報通信学会 第 11 回システム LSI ワークショップポスターセッション, (2007).
2)　Suricata Home Page, http://www.openinfosecfoundation.org/index.php/download-suricata .
3)　Open Information Security Foundaiton, http://www.openinfosecfoundation.org/ .
4)　Emerging Threats, http://www.emergingthreats.net/ .

5) Suricata PacketPipeline, https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Packet_Pipeline .
6) 大山昇吾，藤野毅: パケット分配装置を用いた分散型侵入検知システム，電子情報通信学会 2009 年暗号と情報セキュリティシンポジウム, (2009).
7) Joshua S. White, Thomas Fitzsimmons and Jeanna N. Matthews: Quantitative Analysis of Intrusion Detection Systems: Snort and Suricata, Cyber Sensing 2013, (2013).
8) 笠原博徳，並列処理技術，コロナ社, pp.148-165 (1991).
9) MIT Lincoln Laboratory, http://www.ll.mit.edu/index.html .
10) DARPA Intrusion Detection Data Sets - MIT Lincoln Laboratory, http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html .
11) CDX Data Set, https://www.itoc.usma.edu/research/dataset/ .
12) Information Technology and Operations Center (ITOC), https://www.itoc.usma.edu/ .