情報処理学会論文誌 Vol.60 No.9 1448–1465 (Sep. 2019)

機械学習ベースNIDS構築のための分散処理フレームワーク

多田 竜之介[1,a)] 中村 純哉[1,b)] 大村 廉[1,c)] 小林 良太郎[2,d)]

受付日 2018年11月26日, 採録日 2019年6月11日

概要：ネットワークベース侵入検知システム（Network-based Intrusion Detection System：NIDS）は未知攻撃の検知とスケーラビリティの確保という2つの課題に直面している．これらの課題を解決するために，ネットワーク侵入検知の研究領域においてはスケーラビリティを確保した機械学習ベースのNIDSがいくつか提案されている．しかしながら，既存研究ではバッチ処理を前提としているために通信の発生後即座に検知結果を得られなかったり，ネットワークトラフィックからの特徴抽出を実装していなかったりするなど，NIDSとしては不完全なものである．そこで本論文では，すべての機能をスケーラブルなストリーム処理が可能な仕組みの上に実装した，より実用的な機械学習ベースのNIDSを構築するための分散処理フレームワークを提案する．これにより，利用者が機械学習によってNIDSの分類部分さえ実装すれば，機械学習ベースのNIDSを容易に構築できる．実際に本フレームワークを利用して小規模なクラスタ上にNIDSを構築し，システムのスループットや処理遅延について評価した．評価の結果，小規模なクラスタであっても1 Gbpsのネットワークトラフィックを十分に処理できた．一方で，特定の状況下では性能が大幅に制限されることも明らかになった．

キーワード：機械学習，ネットワークベース侵入検知システム，分散処理

Distributed Processing Framework for Machine Learning-based NIDS Construction

Ryunosuke Tada[1,a)] Junya Nakamura[1,b)] Ren Ohmura[1,c)] Ryotaro Kobayashi[2,d)]

Received: November 26, 2018, Accepted: June 11, 2019

Abstract: A Network-based Intrusion Detection System (NIDS) is faced with two requirements: detection of unknown attacks and scalability for traffic increase. To address the requirements, in the research field of network intrusion detection, researchers have proposed several scalable machine learning-based NIDSs. Nevertheless, they are incomplete as a NIDS for the following reasons: (1) they cannot detect malicious activities in near real time because of their batch processing approach, and (2) they do not implement feature extraction from network traffic. Therefore, in this paper, we propose a distributed processing framework for a more practical machine learning-based NIDS, all the functions of which are implemented with some scalable frameworks dealing with stream processing. Our framework enables users to easily construct machine learning-based NIDS. All you have to do is to implement only classification phase of the NIDS based on machine learning. In this paper, we actually used our framework to construct a NIDS on a small cluster and evaluated its throughput and delay time. The evaluation results show that the NIDS produced by our framework can process above 1 Gbps of network traffic in spite of the small cluster. On the other hand, the performance is severely limited under certain conditions.

Keywords: machine learning, network-based intrusion detection system, distributed processing

[1] 豊橋技術科学大学
Toyohashi University of Technology, Toyohashi, Aichi 441–8580, Japan
[2] 工学院大学
Kogakuin University, Shinjuku, Tokyo 163–8677, Japan

[a)] tada.ryunosuke@usl.cs.tut.ac.jp
[b)] junya@imc.tut.ac.jp
[c)] ren@tut.jp
[d)] ryo.kobayashi@cc.kogakuin.ac.jp

1. はじめに

インターネットを利用したサービスの普及によって，人々の生活はより便利で豊かなものになった．しかしながら，同時にサイバー攻撃による被害も生じており，安全なインターネットの利用が脅かされている．ネットワークベース侵入検知システム（Network-based Intrusion Detection System：NIDS）によるネットワーク監視は，サイバー攻撃防御法の 1 つであるものの，現在 2 つの課題に直面している．

まず 1 つ目の課題は，検知性能の問題である．現在主流の NIDS はシグネチャ型と呼ばれる方式で，既知の攻撃を解析して得られるシグネチャをもとに悪性通信を検知する．しかしながら，既知の攻撃を解析してシグネチャを作成する性質上，未知の攻撃に対処できないという問題がある．

次に 2 つ目の課題は，処理時間の問題である．インターネットトラフィックは年々増加しており，NIDS が監視するネットワークトラフィックも増加していくと予想される [1], [2]．仮に将来的なネットワークトラフィックの増加に対してマシンの性能が追いつかなくなった場合，既存の NIDS で採用されているような単一の高性能なマシンによる監視が行えなくなる問題がある．したがって，未知の攻撃に対処できるだけでなく，将来的なネットワークトラフィックの増加にも対応できるスケーラビリティをも兼ね備えた NIDS が必要である．

このような背景から，ネットワーク侵入検知の研究領域においては機械学習ベースの NIDS についての研究が行われている [3], [4]．機械学習ベースの NIDS では，正常時の通信の様子をモデル化し，モデルから大きく外れた通信を悪性通信と見なして検知する．これにより，既知の攻撃だけでなく未知の攻撃にも対処できると期待されている．

さらに近年の研究においては，機械学習部分を分散処理フレームワークを利用して実装することで，実用を考慮したスケーラブルなシステムを提案したものも登場している [5], [6], [7], [8], [9], [10]．しかしながら，スケーラブルな機械学習ベースの NIDS についての既存研究ではいくつかの問題がある．

まず，内部でバッチ処理を前提とした分散処理フレームワークを利用している問題がある．ネットワークトラフィックをバッチ処理する場合，ネットワークトラフィックを適切に分割して始点と終点を決定できることと，該当区間のネットワークトラフィックをすべて保持できることが必要である．しかしながら，ネットワークトラフィックは途切れることなく発生し続けるうえに流量も変化するストリームデータであるため，始点と終点を決定できないうえにデータ量も予測できない．加えて，仮にバッチ処理が行えたとしても，バッチ処理の開始が始点のデータの発生から大きく遅れてしまう恐れがある．

次に，ネットワークトラフィックからの特徴抽出を実装していないという問題がある．一部の既存研究におけるシステムでは，入力としてネットワークトラフィックからの特徴抽出済みのテキストファイルを必要とするものがある．このようなシステムを利用するためには，利用者がネットワークトラフィックからの特徴抽出部分を実装しなければならない．

そして，スループットや処理遅延といった指標でシステムを評価していないという問題がある．より実用的な NIDS を構築するためには，どの程度の流量のネットワークトラフィックを処理できるかという点や，通信の発生から検知結果が得られるまでの処理遅延がどの程度かという点でも評価する必要がある．

最後に，通信内容の保存や利用者への要約の提示といった，データの管理方法を考慮していないという問題がある．NIDS の性質上，NIDS が悪性通信を検知してからの実際の対応はネットワーク管理者に委ねられる．この際，過去にどのような通信が行われたかを把握する必要があるため，監視対象のネットワークにおける通信内容を保存しておいて分析できることや，分析結果に基づいて要約するような機能が必要である．

このように，既存研究で提案されたシステムは NIDS として不完全なものである．NIDS を構築するうえでは，ネットワークトラフィックがストリームデータであることと，通信の発生後即座に検知結果が得られる必要があることをふまえて処理しなければならない．このため，より実用的な NIDS を構築するためには，連続して発生するデータをそのつど処理できるストリーム処理が適している．加えて，スループットや処理遅延の測定と，通信内容を含む分類結果の保存と要約機能も必要である．

そこで本論文では，すべての機能をスケーラブルなストリーム処理が可能な仕組みの上に実装した，より実用的な機械学習ベースの NIDS を構築するための分散処理フレームワークを提案する．すべての機能をスケーラブルなストリーム処理によって実装するため，通信の発生後即座に検知結果を得られるだけでなく，将来的なネットワークトラフィックの増加にも対応できる．本論文においては，機械学習ベースの NIDS に必要な機能を次のように定義する．

(1) ネットワークトラフィックからの特徴抽出
(2) 特徴量に基づく機械学習による分類
(3) 分類結果の保存や要約

本フレームワークでは，前述のうち機械学習による分類部分を除いたすべての機能を提供する．このため，利用者は機械学習による分類部分のみを実装すれば，容易に実用的な機械学習ベースの NIDS を構築できる．

本論文では，提案するフレームワークの詳細を述べるとともに，実際にフレームワークを利用してシステムを構築し，システム全体のスループットや処理遅延といった評価

指標についても測定する．これにより，本フレームワークを利用してシステムを構築する場合の，ハードウェア性能とシステム性能の関係も明らかにする．

以下，2 章では関連研究について述べる．3 章では本フレームワークの概要について述べ，4 章では具体的な実装について述べる．5 章では，実際に本フレームワークを利用してシステムを構築し，ネットワークトラフィックを処理した場合のスループットや処理遅延などを評価する．6 章では，本フレームワークを利用したシステムの理論上の性能限界について考察する．7 章では，機械学習による分類部分において，既存研究で利用できる特徴量と本フレームワークで利用できる特徴量との差異が分類性能に与える影響について評価する．8 章はまとめである．

2. 関連研究

機械学習ベースの NIDS については，これまでに多くの研究が行われてきた [3], [4]．さらに近年では，実用を考慮して分散処理フレームワークを利用したスケーラブルな機械学習ベースの NIDS を提案した研究も登場している [5], [6], [7], [8], [9], [10]．以降では，機械学習ベースの NIDS についての既存研究の概要と，スケーラブルな機械学習ベースの NIDS についての既存研究について述べる．

2.1 機械学習ベースの NIDS についての既存研究の概要

これまでの機械学習ベースの NIDS についての研究においては，ネットワークトラフィックから何らかの方法でセッションを構築・集約して得られるセッションベース特徴量に基づき，機械学習を利用した悪性通信の検知が試みられてきた [4]．これらの研究は，文献 [11] や文献 [12] のように，単一の数十次元程度の情報を持つサンプル（セッションベース特徴量）に対して機械学習による分類を適用したものであった．したがって，このような単一のセッションベース特徴量に基づいて悪性通信を検知する既存研究においては，機械学習部分で利用される手法はネットワーク侵入検知に特化したものではなく，汎用的な機械学習手法をネットワーク侵入検知に適用したものであった．

機械学習による悪性通信の検知性能の評価のためには，既存研究においては NIDS 評価用データセットとして公開されているいくつかのデータセットが利用されてきた．具体的には，KDD Cup 1999 Data [13] や NSL-KDD dataset [14]，Kyoto Dataset [15], [16], [17]，UNSW-NB15 data set [18], [19] といったデータセットである．これらのデータセットは，すべてセッションベース特徴量を採用したものであり，機械学習ベースの NIDS についての近年の研究においても評価のために利用されている [4]．

Gao らは，アンサンブル学習に基づく新たな半教師あり学習手法をネットワーク侵入検知向けに提案した [11]．まず，ラベルありのデータを利用して分類器を作成し，作成した分類器によってラベルなしデータに正解をつける．次に，ラベルなしのデータを利用して分類器を作成する際には，“Fuzziness-based Method” によって分類器の作成に利用するサンプルを選別する．分類部分は，複数の CART 分類器による分類結果を 3 層のニューラルネットによって重み付けして最終結果とする設計である．NSL-KDD dataset を利用した評価では，単一の機械学習手法を利用した場合と比較して提案手法がより高い正解率を達成できることを示した．さらに，他の既存研究と比較しても優れており，提案手法がネットワークベースの侵入検知に貢献できることを示した．

Al-Qatf らは，Sparse Autoencoder（SAE）と Support Vector Machine（SVM）を組み合わせて Self-taught Learning based Intrusion Detection System（STL-IDS）を提案した [12]．提案手法によって効率的な次元削減が可能になり，学習時と評価時の両方で処理時間を短縮できるだけでなく，いくつかの評価指標における分類性能の向上も期待できる．NSL-KDD dataset を利用した評価では，単体の SVM と比較して学習時と評価時の両方で処理時間を短縮できるだけでなく，正解率や精度，再現率や F 値の面でも優れていることを示した．さらに，将来的には提案手法を並列処理プラットフォームや GPU を利用して実装することで，さらなる学習時および評価時の処理時間の短縮が可能であることも示した．

2.2 スケーラブルな機械学習ベースの NIDS についての既存研究

近年では，年々増加するネットワークトラフィックに対応するために，分散処理フレームワークを利用したスケーラブルな機械学習ベースの NIDS についての研究も登場している．

Kato らは，Apache Hadoop と Apache Hive，Apache Spark を組み合わせた実用的なアノマリ型 NIDS を提案した [5]．Apache Hadoop のライブラリである hadoop-pcap-lib を利用して PCAP ファイルの情報を Apache Hive の Hive テーブルに格納し，Apache Spark によって特徴抽出や正規化，機械学習による検知を行う仕組みである．実際に 90.9 GB の PCAP ファイルを処理し，著者らの以前のアプローチと比較してより高速に解析できることを示した．しかしながら，パケットの解析に利用された Apache Hadoop はバッチ処理を前提としたフレームワークである．このため，バッチ処理のタイミングによっては，通信の発生から検知結果が得られるまでに大きな遅延が生じる恐れがある．

Manzoor らは，Apache Storm を利用した機械学習ベースのリアルタイム NIDS を提案した [6]．KDD Cup 1999 Data を入力として，Apache Storm を利用して正規化や Support Vector Machine による分類を実装した．Apache Storm はストリーム処理フレームワークであり，通信の発

表 1 フレームワークで利用できるセッションベース特徴量の一覧
Table 1 List of session-based features available on the framework.

属性名	概要
	基本特徴量（16 種類）
Timestamp	セッションの開始時刻（ミリ秒単位の UNIX 時間）
Duration	セッションの継続時間（秒）
Source IP Address	送信元 IP アドレス
Source Port Number	送信元ポート番号
Destination IP Address	宛先 IP アドレス
Destination Port Number	宛先ポート番号
Protocol	プロトコル（TCP・UDP・ICMP のいずれか）
Service Type	サービス種類（http・smtp など）
Connection State	セッション終了時の状態（正常終了・接続拒否など）
Direction	通信の方向（L2L・L2R・R2L・R2R のいずれか）
Source Packets	送信パケット数
Source Bytes	ペイロードのみの送信バイト数
Source IP Bytes	ヘッダを含んだ送信バイト数
Destination Packets	受信パケット数
Destination Bytes	ペイロードのみの受信バイト数
Destination IP Bytes	ヘッダを含んだ受信バイト数
	ホストベース特徴量（5 種類）
Dst Host Count	宛先アドレスが同じ過去 100 セッションのうち現在のセッションと送信元アドレスも同じ数
Dst Host Same Src Port Count	Dst Host Count 特徴で該当したセッションのうち現在のセッションと送信元ポートも同じ数
Dst Host Serror Count	Dst Host Count 特徴で該当したセッションのうち “SYN” エラーが起こった数
Dst Host Srv Count	宛先アドレスが同じ過去 100 セッションのうち現在のセッションとサービス種類も同じ数
Dst Host Srv Serror Count	Dst Host Srv Count 特徴で該当したセッションのうち “SYN” エラーが起こった数

生後即座に処理できるため NIDS の実装に適している．しかしながら，入力として KDD Cup 1999 Data のような特徴抽出済みのテキストファイルを必要とする設計であるため，ネットワークトラフィックからの特徴抽出部分を別に実装する必要がある．このため，ネットワークトラフィックから監視する場合，提案手法単体では機械学習ベースの NIDS を構築できない．

Dahiya らは，Apache Spark を利用して機械学習ベースのネットワーク侵入検知機構を実装し，大規模なデータセットにおいて性能を評価した [7]．2 種類の次元削減手法と複数の機械学習アルゴリズムからそれぞれ 1 つずつを選択し，すべての組合せで検知性能を評価した．評価時には，KDD Cup 1999 Data では現在のネットワーク環境を反映できていないことから，より現実に即したデータセットとして UNSW-NB15 data set を利用した．しかしながら，こちらについても入力として特徴抽出済みのテキストファイルを必要とする設計のため，ネットワークトラフィックからの特徴抽出部分は別に実装する必要がある．

このほかにも，分散処理フレームワークを利用した機械学習ベースの NIDS についてはいくつかの研究が存在する [8], [9], [10]．しかしながら，いずれにおいても Apache Hadoop のようなバッチ処理を前提とした分散処理フレームワークを利用しているか，あるいは入力として特徴抽出済みのデータが得られることを前提としており，機械学習ベースの NIDS としては不完全である．特に，ネットワークトラフィックからストリーム処理で特徴量を抽出してその後の処理を連携させた例は我々の知る限り存在しない．

3. フレームワークの概要

本フレームワークにおいてネットワークトラフィックから抽出する特徴量と，処理の流れについて詳細を述べる．

3.1 フレームワークで利用できる特徴量

本フレームワークを設計するうえで，機械学習部分は 2.1 節で述べたようなセッションベースの特徴量を利用することを前提とする．このようにするのは，2.1 節で述べたように，セッションベースの特徴量を利用した機械学習ベースの NIDS についての研究が多く存在し，これまでの研究成果をそのまま活用できるためである．

本フレームワークで利用できる特徴量は，IP アドレスやポート番号といったセッションの基本的な情報と，複数のセッションを特定の条件で集約して得られる特徴量である．現在のところ，本フレームワークで利用できる特徴量は**表 1** に示す 21 種類である．表 1 に示した特徴量は，KDD Cup 1999 Data に含まれる “basic features” を拡張した 16 種類の基本特徴量と，“host-based traffic features” の一部にあたる 5 種類のホストベース特徴量である．host-based traffic features とは，抽出対象のセッションと宛先アドレ

スが同じ過去の 100 セッションについて集計して得られる特徴量である．以降では，説明のために，表 1 に示した 16 種類の基本特徴量を「基本特徴量」，基本特徴量に表 1 の 5 種類のホストベース特徴量を加えた 21 種類の特徴量を「完全な特徴量」と表記して区別する．

KDD Cup 1999 Data には 41 種類の特徴量が含まれているものの，Kyoto 2006+ Dataset の作成時に述べられたように，実質的に冗長であったりプロトコル依存の情報を含む特徴量も含まれている [15]．このため，本フレームワークにおいても Kyoto Dataset に含まれない KDD Cup 1999 Data の特徴量を除外した．加えて，過去 2 秒間の通信すべてについて集計して得られる “time-based traffic features” についても，大規模ネットワークにおいて集計対象のセッション数が多くなりすぎる恐れがあることと，すべての通信をまとめて扱う必要がありスケーリングの妨げになることから除外した．

なお，詳細は 7 章で述べるものの，本フレームワークにおいて除外したセッションベース特徴量が既存研究の分類性能に与える影響は軽微である．

3.2 フレームワークの処理の流れ

本フレームワークにおける処理の流れを**図 1** に示す．図 1 に示したように，まずネットワークトラフィック（パケット）からセッションを構築し，セッションの終了時に IP アドレスやポート番号といった表 1 の基本特徴量を取得する．次に，基本特徴量を特定の条件で集約して表 1 のホストベース特徴量を抽出し，基本特徴量にホストベース特徴量を加えて完全な特徴量を得る．最後に，抽出した完全な特徴量から利用者による任意の実装で機械学習による分類を実行し，完全な特徴量に分類結果を加えて本フレームワーク規定のフォーマットで永続データストアに格納する．格納したデータは本フレームワークがあらかじめ定義した要約ビューの作成に利用するほか，より高度な分析が必要な場合は利用者による任意の分析に利用できる．

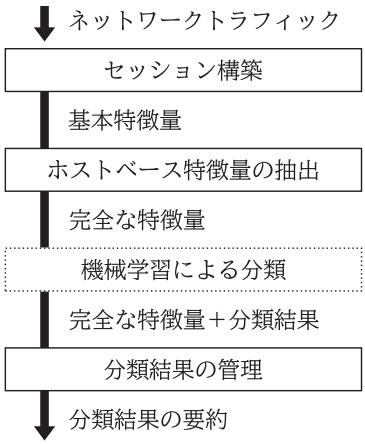

図 1 フレームワークの処理の流れ
Fig. 1 Process flow of the framework.

4. フレームワークの実装

3 章で述べた設計に基づき，詳細な実装について述べる．本フレームワークにおける実際の処理の流れと構成を**図 2** に示す．図 2 に示したように，ストリーム処理が可能な既存の分散処理フレームワークを組み合わせて全体のシステムを構築しており，一部制約はあるもののすべての処理はスケーラブルである．以降では段階ごとに実装の詳細を述べる．

4.1 Apache Kafka によるメッセージキューイング

Apache Kafka [20] は分散型のストリーミングプラットフォームであり，大規模なストリームデータの保存と配信機能を提供する．Apache Kafka を利用することで，リアルタイムストリーム処理を必要とするようなアプリケーションを実現できる．

Apache Kafka では，データ（メッセージ）のやりとりに Publisher/Subscriber パターンを採用している．Publisher/Subscriber パターンでは，次のようにメッセージをやりとりする．

(1) Publisher（Producer）が Broker の特定の Topic（メッセージが属するグループのようなもの）にメッセージを送信する．
(2) メッセージが追加された Topic をあらかじめ購読していた Subscriber（Consumer）にメッセージを配信する．

加えて Apache Kafka では，Consumer へのメッセージ配信に Pull 型を採用しており，メッセージを受信するタイミングを Consumer 側が決定できる．このように，Apache Kafka を利用するアプリケーションの側から見れば，Apache Kafka はメッセージキューのように振る舞う．

また，Apache Kafka では Topic をさらに任意の数の Partition という単位に分けて管理している．

メッセージの送信時には，基本的に Partition を意識す

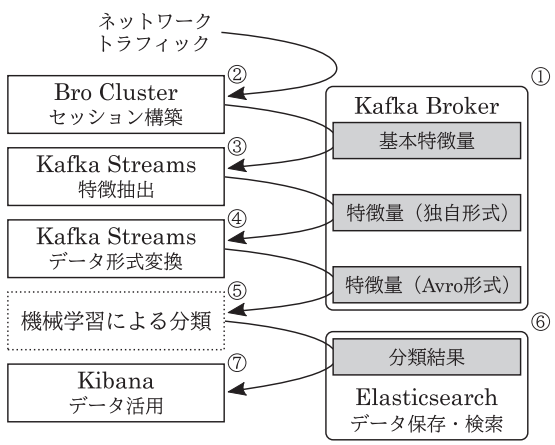

図 2 フレームワークの実装の詳細
Fig. 2 Implementation details of the framework.

る必要はなく，メッセージがどの Partition に保存されるかはラウンドロビンで決定される．ただし，メッセージを特定の Partition に保存したい場合にはメッセージにキーを指定することで実現できる．メッセージにキーが設定されている場合，メッセージがどの Partition に保存されるかはキーのハッシュ値に基づいて決定される．なお，詳細は 4.2 節で述べるものの，本フレームワークにおいてはホストベース特徴量の抽出のためにこの機能を利用する．

メッセージの受信時には，Partition と Consumer の関係に注意しなければならない．特定の Topic を購読する場合，Apache Kafka では Consumer Group（同じ ID が設定された Consumer のグループ）のすべての Consumer に対して Partition 単位でメッセージが振り分けられる．したがって，Topic ごとの最大並列数は Partition の数によって決まる．また，Consumer の数と Partition の数を適切に決定しなければ Consumer の負荷が偏ってしまう恐れがある．

本フレームワークにおいては，図 2①に示したように，Apache Kafka を一時的なデータストアとして利用しており，ネットワークトラフィックからの特徴抽出を行う過程で生じる中間データを保存している．また，完全な特徴量を利用者側のアプリケーションに提供する際には，Apache Kafka が入力部分として振る舞う．このため，Topic の Partition 数を利用者が適切に設定する必要がある．

4.2 Bro Cluster によるセッション構築

ネットワークトラフィックからセッションを構築するために，図 2②に示したように，本フレームワークでは Bro IDS [21] を利用する．

Bro IDS はネットワーク監視ツールであり，オープンソースで公開されている．シグネチャベースのネットワーク監視とは異なり，ネットワークトラフィックをセッション単位で扱って監視するアプローチを採用している．セッション単位での監視にはアプリケーションレイヤでの監視を行うための複数の解析器が利用でき，高度なネットワーク監視が可能である．また，独自のスクリプト言語である Bro Scripts によってプログラムの挙動を変更できるため，利用者の好みに応じて柔軟な処理が可能である．

本フレームワークでは，Bro IDS に含まれる Bro Cluster という仕組みを利用する．Bro IDS は単体でスケーラブルな設計ではないため，Bro Cluster では複数のプロセス・マシンで Bro IDS を並列実行する仕組みを提供することでスケーラビリティを確保する．内部では，各 Bro IDS プロセスに対してネットワークトラフィックを適切に分配することで並列処理を可能にしている．ただし，プロセス間の分配は Bro Cluster 側で扱うものの，マシン間の分配は扱えない．本フレームワークにおいてもマシン間の分配は扱えないため，複数のマシンで Bro Cluster を構築する場合は利用者による適切なネットワークトラフィックの分配が必要である．

Bro IDS によるセッションの構築後（セッションの終了時），得られたセッション情報を基本特徴量として Kafka Broker へ書き込む．Bro IDS から Kafka Broker への基本特徴量の書き込みには，Bro IDS のプラグインに含まれる bro-kafka-plugin が利用できる．しかしながら，bro-kafka-plugin の仕様上，基本特徴量を JSON 形式に変換してから Kafka Broker へ書き込んでしまう．JSON 形式で基本特徴量を扱う場合，Kafka Broker を介してデータを変換していく本フレームワークでは 2 つの問題が生じる．

まず，1 つ目の問題はデータサイズである．ネットワークを介してデータをやりとりするうえでは，可能な限りデータサイズを小さくする必要がある．しかしながら，JSON 形式ではすべてのデータを単一の文字列に変換して記録するため，多くの場合でメモリ上のデータサイズよりも大きな領域を必要としてしまう．さらに，bro-kafka-plugin では基本特徴量をキー（属性名）と値のペアで記録するため，キーの分だけ余分なデータを含んでしまう．

次に，2 つ目の問題はシリアライズ・デシリアライズ時の計算コストである．本フレームワークにおいては，データを変換する過程で複数回のシリアライズとデシリアライズが必要なため，それぞれの処理は高速でなければならない．しかしながら，JSON 形式ではすべてのデータを単一の文字列に変換してしまうため，特にデシリアライズ時に計算コストが高くなってしまう．

このような問題から，本フレームワークにおいては独自のデータ形式を中間データ形式として採用した．データ形式の概要を**表 2** に示す．表 2 に示したように，すべてのデータをメモリ上のデータ形式のまま扱い，IP アドレスや数値については可能な限り小さな領域を使うように設計した．このようにすることで，データサイズを小さくできるだけでなく，シリアライズとデシリアライズに必要な計算コストも抑えられる．本フレームワークでは，

表 2 中間データ交換用データフォーマット
Table 2 Data format for exchanging intermediate data.

属性名	領域（Byte）
ヘッダ（接続状態，プロトコル，方向など）	4
送信元・宛先ポート番号	4
タイムスタンプ	8
ログ作成時刻	8
サービス種類（非 null 時）	～31
送信元・宛先 IP アドレス	8～32
セッション継続時間（非 null 時）	8
送信側統計情報（パケット数・バイト数）	6～24
受信側統計情報（パケット数・バイト数）	6～24
ホストベース特徴量（非 null 時）	10
合計	44～153

bro-kafka-plugin を一部変更して独自形式で基本特徴量を Kafka Broker へ書き込めるようにした．

なお，独自形式の採用によるデータサイズおよびシリアライズ時とデシリアライズ時の処理時間の改善については付録 A.1 章で述べる．

また，Kafka Broker への基本特徴量の書き込み時には，セッションの宛先アドレスを Kafka メッセージのキーとして設定する．このようにすることで，宛先アドレスごとに基本特徴量が保存される Partition を固定できるため表 1 に示したホストベース特徴量を抽出できる．

4.3 Kafka Streams によるホストベース特徴量の抽出

Bro IDS が構築した基本特徴量は Kafka Broker に保存されている．次の段階では，基本特徴量から表 1 に示したホストベース特徴量を抽出する．本フレームワークにおいては，図 2③に示したように，Kafka Broker 上の基本特徴量からのホストベース特徴量の抽出に Kafka Streams を利用する．

Kafka Streams は，Apache Kafka に内包されているリアルタイムストリーム処理を実現する仕組みである．Kafka Broker の任意の Topic からメッセージを読み込み，任意の処理を行って結果を任意の Topic に書き込める．Kafka Streams を利用するアプリケーション側では，Kafka Streams からメッセージごとにキーと値のペアが与えられ，任意の処理を行って任意の型でキーと値のペアを返す設計になっている．Kafka Streams を利用して作成したアプリケーションはスケーラブルであるほか，exactly once 処理が可能（標準では at least once）で耐障害性も備えている．

本フレームワークにおいては，受け取ったキーと値を任意の型のキーと値に変換できる Kafka Streams の Transformer クラスを継承し，基本特徴量からホストベース特徴量を抽出する独自クラスを作成した．独自クラスでは，Kafka Streams から宛先アドレス（キー）と基本特徴量（値）のペアが与えられ，null（キー）と完全な特徴量（値）を返す処理を行う．完全な特徴量を Kafka Broker へ送信する際には Kafka メッセージにキーを設定しないため，ランドロビンによりメッセージを保存する Partition が決定される．したがって，特徴抽出の都合上宛先アドレスを Kafka メッセージのキーにしていた基本特徴量の場合とは異なり，完全な特徴量はすべての Partition へ均等に分配される．

4.4 Kafka Streams による完全な特徴量の Avro 形式への変換

完全な特徴量は，表 2 に示した独自形式で Kafka Broker に保存されている．しかしながら，完全な特徴量を利用者側のアプリケーションに提供することを考えると，独自形式のデータでは扱いにくい恐れがある．このため，本フレームワークにおいては独自形式の完全な特徴量を Avro 形式 [22] に変換して利用者側のアプリケーションに提供する．

Avro 形式とは，Apache Avro [22] において定義されているデータシリアライゼーション形式である．あらかじめ定義したスキーマに基づいてデータをやりとりできるだけでなく，データ容量と計算コストの面で優れたバイナリ形式のデータ形式である．

本フレームワークにおいては，図 2④に示したように，Kafka Streams を利用して完全な特徴量を独自形式から Avro 形式へ変換する．Kafka Streams 用の，独自形式から完全な特徴量オブジェクトへの Deserializer クラスと，完全な特徴量オブジェクトから Avro 形式への Serializer クラスを実装することで，完全な特徴量を Avro 形式に変換して Kafka Broker へ書き込める．

4.5 完全な特徴量に基づく機械学習による分類

図 2⑤に示したように，Kafka Broker に Avro 形式で保存されている完全な特徴量に基づき，機械学習を利用してセッションが悪性かどうかを分類する．しかしながら，機械学習による分類を行う場合，正規化や特徴選択，次元削減といった前処理や，その後の分類部分でどのような手法を利用するかという点において無数の選択肢が存在する．このため，現在のところ本フレームワークにおいては機械学習による分類部分は扱わずに利用者側に実装を任せることにした．以降では，利用者側でアプリケーションを作成する場合の，機械学習部分の仕様や注意点について詳細を述べる．

4.5.1 機械学習による分類部分で利用できる特徴量

利用者側のアプリケーションでは，機械学習による分類のために表 1 に示した 21 種類の特徴量を利用できる．加えて，メッセージサイズが大きくなるため Kafka メッセージには含めていないものの，既存の特徴量を組み合わせることで新たな特徴量も利用できる．たとえば，送信バイト数（Source IP Bytes）をセッション継続時間（Duration）で割ることで，UNSW-NB15 data set で特徴量として含まれる 1 秒ごとの平均送信バイト数（sload）を得られる．他にも，セッション中の平均送信/受信パケットサイズや 1 秒ごとの平均送信/受信パケット数，すべての通信量に占める送信量の割合なども抽出でき，悪性通信の分類に利用できる可能性がある．

4.5.2 機械学習による分類部分の実装方法

本フレームワークからは，セッションの終了時に Avro 形式の完全な特徴量を利用者側のアプリケーションに提供する．本フレームワークから Avro 形式で完全な特徴量を受け取ったら，Avro 形式のデータを解釈して完全な特徴量に基づく機械学習による分類処理を行う．機械学習による分類部分には，2.1 節で述べたような，単一のセッションベース特徴量に基づいてセッションが悪性かどうかを分

類する任意の汎用的な機械学習手法を利用できる．

利用者側でアプリケーションを実装するにあたり，1 章で述べた機械学習ベースの NIDS の要件を満たすためには，機械学習による分類部分をスケーラブルなストリーム処理が可能な仕組みの上に実装しなければならない．加えて，完全な特徴量は Kafka Broker に保存されているため，Kafka Broker からの読み込みにも対応しなければならない．

幸いなことに，Kafka Broker からの読み込みが可能な分散型のストリーム処理フレームワークが数多く公開されている．本フレームワークでも利用している Kafka Streams はもちろんのこと，Apache Spark [23] の Spark Streaming や Apache Storm [24]，Apache Samza [25]，Apache Flink [26] などが利用できる．これらの既存の分散処理フレームワークを利用する場合，利用者側のアプリケーションは基本的に Java で実装する必要がある．

あるいは，Apache Kafka の Java Client や，C/C++ の Apache Kafka Client ライブラリである librdkafka [27] を利用してすべてを利用者側で実装することもできる．librdkafka を利用して利用者側のアプリケーションを実装する場合は，librdkafka に各プログラミング言語用のバインディングが用意されているため，任意のプログラミング言語で実装できる．

なお，機械学習による分類処理の具体的な実装方法については，ストリーム処理フレームワーク上に特徴量に基づいて機械学習による分類を行う一般的なアプリケーションを実装する場合と同様であるため本論文では割愛する．

4.5.3 機械学習による分類部分からの出力

機械学習によるセッションの分類結果が得られたら，分類結果を本フレームワーク規定の属性で記録し，完全な特徴量と合わせて JSON 形式で 4.6 節で述べる Elasticsearch [28] に送信する必要がある．機械学習による分類部分を 4.5.2 項で述べたような既存のストリーム処理フレームワークを利用して実装した場合には，各フレームワークが Elasticsearch へのデータ送信に標準で対応している場合があり容易に実装できる．

分類結果を記録するための規定の属性は**表 3** のとおりである．表 3 に示したように，本フレームワークにおいてはセッションが悪性かどうかを示す is_malicious 属性さえ得られればよい．それ以外の属性について分類結果が得られるかどうかは利用者側の実装に依存するものの，得られる場合は記録しておくことでその後の分析に活用できる．

なお，Elasticsearch はスキーマレスであるため，分類結果を送信する際に利用者が任意のデータを追加して保存することもできる．

表 3 分類結果のデータフォーマット
Table 3 Data format of classification results.

属性名	データ型	概要	必須
is_malicious	Bool	セッションが悪性かどうか	○
probability	Float	分類の確度	
severity	Float	悪性通信の深刻度	
category	String	悪性通信のカテゴリ名（例：DoS）	
sub_category	String	悪性通信のサブカテゴリ名（例：SYN flood）	
description	String	悪性通信の詳細な説明	

4.6 Elasticsearch による分類結果の保存と Kibana による可視化

完全な特徴量と分類結果を永続化するために，図 2 ⑥に示したように，本フレームワークにおいては Elasticsearch を利用する．Elasticsearch はデータの永続化と，保存したデータに対する検索と分析が可能な分散型のシステムである．地理情報や構造化データといった複雑なデータ型が利用できたり，スキーマレスでデータを投入できたりする特長を備えながらも，柔軟で高速な検索と分析が可能なうえにスケーラブルである．

完全な特徴量を Elasticsearch に保存することにより，利用者は監視対象のネットワークで発生したすべてのセッション情報に対して高速な検索と分析が可能である．これにより，機械学習によるネットワーク監視に加えて，利用者による任意のネットワーク監視にも貢献できる．

さらに，図 2 ⑦に示したように，本フレームワークにおいてはデータの可視化のために Kibana [29] を利用する．Kibana は Elasticsearch のためのデータ可視化ツールである．Elasticsearch の強力な検索機能を利用して，任意の値を表示したり，線グラフや円グラフを作成したりできる．また，地理情報を持ったデータに対しては，ヒートマップを作成して地図上に表示するといったこともできる．加えて，あらかじめ作成しておいたグラフや地図などを部品化しておくことができ，保存しておいた部品を画面上に自由に配置してダッシュボードを作成することもできる．

本フレームワークにおいては，ネットワークの状態を確認するために有用と思われる**図 3** に示すようなダッシュボードをあらかじめ作成してある．これにより，利用者は追加の操作を必要とせずにネットワークの状態を視覚的に確認できる．もちろん，新たな部品を作成して追加することや，不要な部品を削除することができるため，利用者の必要に応じてダッシュボードを改良できる．

4.7 フレームワークの展開例

本フレームワークを利用したシステムを展開する場合，必要なネットワーク監視能力に応じて柔軟に構成できる．**図 4** に，本フレームワークを利用したシステムの展開例を示す．

たとえば，図 4 (a) に示したように，すべての機能をオ

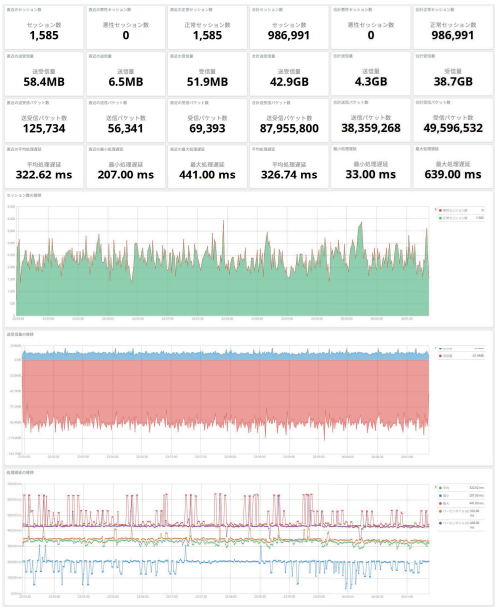

図 3 ネットワーク監視用のダッシュボード
Fig. 3 Dashboard for network monitoring.

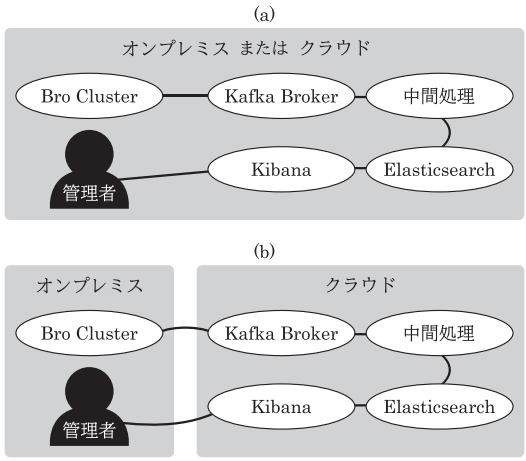

図 4 フレームワークの展開例
Fig. 4 Examples of deployment pattern for the framework.

ンプレミス環境またはクラウド環境のマシン上に展開する構成が可能である．

あるいは，図 4（b）に示したように，ネットワークトラフィックからのセッション構築部分のみをオンプレミス環境のマシン上に展開し，その他すべての機能をパブリッククラウドのようなクラウド環境のマシン上に展開する構成も可能である．基本特徴量を Kafka Broker へ送信する際には，Apache Kafka の機能によって暗号化した通信を利用することでセキュリティを確保できる．また，Elasticsearch と Kibana についても通信を暗号化する機能を持っているため，分析機能や可視化機能も安全に利用できる．

本フレームワークを利用してオンプレミス環境のマシンを保護する場合には，構成（a）か構成（b）を状況に応じて選択できる．

オンプレミス環境で構成（a）を採用する長所は，すべてのマシンがオンプレミス環境に存在するため，高いセキュリティを確保できる点と処理遅延を抑えられる点である．一方で短所は，本フレームワークを展開するためのマシンをすべてオンプレミス環境に用意する必要があるため，ハードウェアの導入や更新，運用に多大なコストが生じる恐れがある点である．

オンプレミス環境で構成（b）採用する長所は，特にパブリッククラウドを利用する場合には，必要なネットワーク監視能力に応じて柔軟にハードウェア構成を変更できるため，ハードウェアの導入や更新，運用に必要なコストを抑制できる可能性がある点である．加えて，Apache Kafka や Elasticsearch には専用のホスティングサービスが存在するため，本フレームワークをより容易に導入・運用できるという利点もある．一方で短所は，通信は暗号化されるものの，自組織の通信内容に関する情報を外部に送信する必要がありセキュリティリスクをともなう点である．加えて，基本特徴量をクラウド環境に転送する必要があるため，専用のネットワーク帯域が必要になるうえに処理遅延が生じる恐れもある．

また，本フレームワークを利用してクラウド環境のマシンを保護する場合には構成（a）が利用できる．ただし，本フレームワークを利用するためには Bro Cluster へのネットワークトラフィックの入力部分でレイヤ 2 での接続が必要なため，環境によっては利用できない場合がある．

なお，図 4 中では機能ごとにマシンを分けてあるものの，実際には複数の機能を単一のマシンに割り当てることも可能であるし，単一の機能を複数のマシンに割り当ててスケーラビリティを確保することもできる．

5. フレームワークの性能評価

本フレームワークを利用して実際にシステムを構築して性能を評価する．しかしながら，図 2 に示したように，本フレームワークを利用してシステムを構築するためには機械学習による分類部分を実装する必要がある．このため本論文では，機械学習による分類は行わずにデータの中継のみを行う仕組みを実装する．具体的には，Avro 形式で保存されている完全な特徴量を Kafka Broker から読み込み，JSON 形式に変換して Elasticsearch に書き込む仕組みである．本論文では，Logstash [30] を利用してこの仕組みを実現する．

Logstash はデータ処理パイプラインとして設計されたもので，様々なデータソースに保存された情報を読み込み，あらかじめ用意された様々なフィルタによって加工した後に様々な出力先に書き込める．Logstash によって，Kafka Broker に保存された Avro 形式の完全な特徴量を読み込み，JSON 形式に変換して Elasticsearch に書き込むという，本フレームワークの機械学習部分に必要な機能を実現できる．ただし，Logstash は標準では Avro 形式のデータ

表 4 性能評価用のマシン仕様の一覧
Table 4 List of machine specifications for performance evaluation.

マシン番号	CPU	動作周波数	論理コア数	RAM	データ用ストレージ	追加 NIC
①	Intel Core i3-3225	3.3 GHz	4	8 GB	HDD 1 TB（SATA3）	Intel EXPI9301CT
②	Intel Core i7-4770K	3.5 GHz	8	16 GB	–	Intel EXPI9402PT
③	Intel Xeon E5-2620 v4×2	2.1 GHz	32	32 GB	HDD 2 TB×3（SATA3）	Intel X540-T2
④	Intel Core i7-4790K	4.0 GHz	8	16 GB	–	Intel X540-T2
⑤	Intel Core i7-4770	3.4 GHz	8	8 GB	–	Intel EXPI9301CT
⑥	Intel Core i7-4770	3.4 GHz	8	8 GB	–	Intel EXPI9301CT
⑦	Intel Core i7-3770K	3.5 GHz	8	16 GB	SSD 250 GB（SATA3）	Intel EXPI9301CT
⑧	Intel Core i7-3770K	3.5 GHz	8	16 GB	SSD 250 GB（SATA3）	Intel EXPI9301CT
⑨	Intel Core i7-2600	3.4 GHz	8	8 GB	–	–

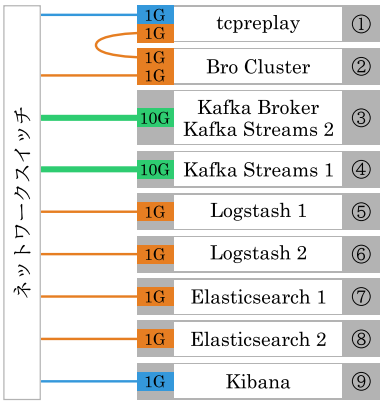

図 5 性能評価用のクラスタ構成
Fig. 5 Cluster configuration for performance evaluation.

の読み込みに対応していないため Avro codec plugin を導入した．

表 4 および**図 5** に，性能評価のために使用したマシンのスペックとクラスタの構成を示す．表 4 に示したように，クラスタを構成するマシンは基本的に一般消費者向けの安価な製品である．図 5 は，クラスタを構成する各マシンの役割とネットワーク接続を表したものであり，各マシンを表す灰色の矩形の右側の数字は表 4 のマシン番号と対応している．橙や緑，青色の線はネットワーク接続を表すもので，橙と青色の接続はリンク速度が 1 Gbps，緑色の接続はリンク速度が 10 Gbps である．また，橙と緑色で示した接続ではマシン側の Network Interface Card（NIC）が Intel 社製であるものの，青色で示した接続では高速な通信が必要ないためマシン標準の NIC である．

本論文では，いくつかの状況を想定したパケットキャプチャファイルから，実際にネットワークトラフィックを発生させて本フレームワークを利用したシステムの性能を評価した．以降では，パケットキャプチャファイルを利用した実験の手順と，いくつかの状況を想定したネットワークトラフィックを処理した場合のシステムの性能評価結果について述べる．

5.1 パケットキャプチャファイルを利用したシステムの性能評価方法

実験の手順と処理の流れは次のとおりである．

まず，マシン ① からパケットキャプチャファイルを Tcpreplay [31]（パケットキャプチャファイルからのパケット再送ツール）によってマシン ② の Bro Cluster 用のネットワークインタフェースへ送信する．Tcpreplay にはパケットキャプチャファイルを任意の速度で送信する機能があり，パケットキャプチャ時の転送量とは異なる速度で実験できる．マシン ② では Bro Cluster 用のネットワークインタフェースで Bro Cluster が待ち受けており，到達したパケットを解析してセッションを構築する．構築したセッションについての情報は，表 2 に示した独自形式に変換してマシン ③ の Kafka Broker 上の基本特徴量用 Topic に送信する．Kafka Broker にはあらかじめ基本特徴量用 Topic を作成してあり，Partition 数は 6 である．

次に，基本特徴量用 Topic にメッセージが追加されたため，マシン ④ で動作しているホストベース特徴量抽出用の Kafka Streams アプリケーションが基本特徴量を受信する．受信した基本特徴量をもとにホストベース特徴量を抽出し，基本特徴量に加えてマシン ③ の Kafka Broker 上の独自形式の完全な特徴量用 Topic（Partition 数は 6）に送信する．マシン ④ ではアプリケーションを 6 スレッドで実行し，基本特徴量用 Topic の Partition と Consumer が 1 対 1 に対応するようにした．

そして，独自形式の完全な特徴量用 Topic にメッセージが追加されたため，マシン ③ で動作しているデータ形式変換用の Kafka Streams アプリケーションが独自形式の完全な特徴量を受信する．受信した独自形式の完全な特徴量を Avro 形式に変換し，マシン ③ の Kafka Broker 上の Avro 形式の完全な特徴量用 Topic（Partition 数は 12）に送信する．マシン ③ でもアプリケーションを 6 スレッドで実行し，独自形式の完全な特徴量用 Topic の Partition と Consumer が 1 対 1 に対応するようにした．

最後に，Avro 形式の完全な特徴量用 Topic にメッセージが追加されたため，マシン ⑤ および ⑥ で動作してい

るメッセージ中継用の Logstash が Avro 形式の完全な特徴量を受信する．受信した Avro 形式の完全な特徴量を，分類結果を付加したと見なして JSON 形式に変換してマシン ⑦ および ⑧ で動作している Elasticsearch に送信する．マシン ⑤ および ⑥ ではそれぞれで Logstash による中継部分を 6 スレッド（合計 12 スレッド）で実行し，Avro 形式の完全な特徴量用 Topic の Partition と Consumer が 1 対 1 に対応するようにした．

このような仕組みによって，ネットワークトラフィックからの特徴抽出から分類結果の永続化までを実際に実行してシステムの性能を評価できる．システムの性能評価にあたっては，システム全体の平均スループットと処理遅延（平均・最大・80 パーセンタイル・90 パーセンタイル），各マシンの平均 CPU 使用率を測定した．

本論文における各指標の定義は，まず，スループットについては処理の終点における 1 秒あたりの処理件数とした．本フレームワークにおける処理は，Elasticsearch による分類結果の永続化をもって完了する．したがって，本フレームワークにおいては Elasticsearch における分類結果の 1 秒あたりの挿入数（処理件数）をスループットとした．

次に，処理遅延については処理の始点から終点までの経過時間とした．本フレームワークにおける処理は，Bro Cluster によるセッション構築に始まり，Elasticsearch における分類結果の永続化をもって完了する．したがって，本フレームワークにおいては，Bro Cluster においてセッション構築が完了してから Elasticsearch において分類結果を挿入するまで（およそ図 2 ②の後から図 2 ⑥の前まで）の時間を処理遅延とした．

最後に，CPU 使用率についてはすべての CPU コアごとの非アイドル割合の平均値とした．マシン単位での CPU 使用率の測定ではあるものの，本論文の実験においてはすべてのマシンが本フレームワークの実行専用に用意したものであるため，本フレームワークに関連しないプロセスの CPU 使用が与える影響は軽微であると考えている．

5.2 現実的なトラフィックを利用した性能評価

現実的な環境におけるネットワークトラフィックを処理した場合の性能を評価した．現実的な環境を再現するために，UNSW-NB15 data set に含まれる PCAP ファイルを利用した．UNSW-NB15 data set に含まれる PCAP ファイルは，実験用ネットワーク上に現実的な正常・悪性通信を発生させられる仕組みを構築し，実際に十数時間にわたって擬似的に通信を発生させて作成したものである [18], [19]．正常通信に加えて悪性通信も含んでいるため，NIDS を評価するうえでより現実的なネットワークトラフィックである．

UNSW-NB15 data set では，条件の異なる 2 種類の PCAP ファイルを利用できる．1 つ目は，2015 年 1 月 22 日の実験を記録したおよそ 50 GB の PCAP ファイルであり，1 秒間に 1 回の悪性通信を含んでいる．2 つ目は，2015 年 2 月 17 日の実験を記録したおよそ 50 GB の PCAP ファイルであり，1 秒間に 10 回の悪性通信を含んでいる．PCAP ファイルは UNSW-NB15 data set のダウンロードページから取得できるものの，それぞれの PCAP ファイルは分割して公開されている．このため，パケット解析ツールである Wireshark [32] に含まれる mergecap を利用し，分割されている PCAP ファイルを結合して単一の PCAP ファイルとして利用した．以降では，2015 年 1 月 22 日分の結合後の PCAP ファイルを Capture 1，2015 年 2 月 17 日の結合後の PCAP ファイルを Capture 2 と表記する．**表 5** に Capture 1 および Capture 2 の詳細を示す．表 5 から分かるように，いずれも 9,000 万を超えるパケットが記録されたものであり，平均パケットサイズは 500 Byte 程度である．

表 5 UNSW-NB15 data set に含まれる PCAP ファイルの詳細
Table 5 Details of PCAP files included in UNSW-NB15 data set.

	Capture 1	Capture 2
キャプチャ時間（sec）	45,347	43,557
パケット数	94,571,342	92,559,914
平均パケットサイズ（Byte）	539	550
平均転送量（kbps）	8,984	9,354

表 6 Capture 1 および Capture 2 の送信結果
Table 6 Transmisson results of Capture 1 and Capture 2.

	Capture 1	Capture 2
送信パケット数	94,571,342	92,559,914
送信量（GB）	47.43	47.43
送信時間（sec）	428.19	427.81
平均送信パケット数（pps）	220,861.48	216,356.01
平均送信量（Mbps）	951.49	952.37

Capture 1 および Capture 2 をそれぞれおよそ 1 Gbps で送信する実験を行った．Capture 1 および Capture 2 を Tcpreplay により送信した結果を**表 6** に示す．表 6 に示したように，Capture 1 および Capture 2 に含まれるすべてのパケットを損失なくおよそ 430 秒で送信した．いずれの場合でも 1 秒ごとの平均送信量はおよそ 950 Mbps であり，1 Gbps でリンクしたマシン間の通信としては上限に近い速度であった．

図 6，**図 7**，**図 8** に，それぞれ Capture 1 送信時のスループット，CPU 使用率，処理遅延を示す．図 6 から分かるように，Capture 1 をおよそ 950 Mbps で送信した場合でもセッション単位で扱えば 2,500 session/sec 程度の流量であった．さらに図 7 から分かるように，Bro Cluster が動作しているマシン ② では CPU 使用率が 30%程度まで上昇したものの，それ以外のマシンでは 10%未満であっ

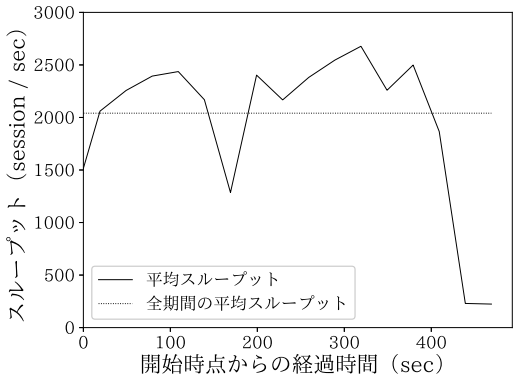

図 6 実験中のスループットの推移（Capture 1）
Fig. 6 Transition of throughput during experiment (Capture 1).

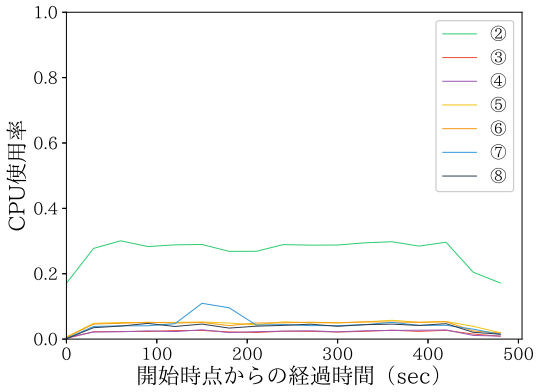

図 7 実験中の CPU 使用率の推移（Capture 1）
Fig. 7 Transition of CPU usage during experiment (Capture 1).

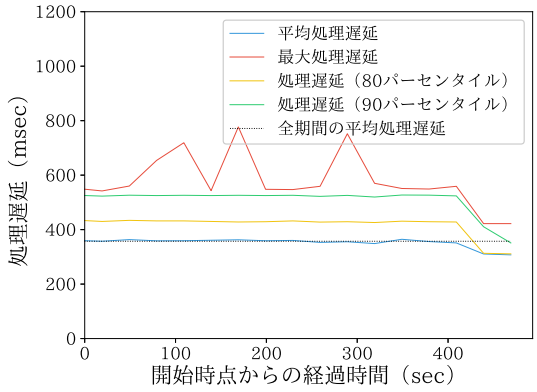

図 8 実験中の処理遅延の推移（Capture 1）
Fig. 8 Transition of lag during experiment (Capture 1).

た．したがって，UNSW-NB15 data set のような現実的なネットワークトラフィックにおいては，本実験のような小規模なクラスタであっても十分に余裕を持って 1 Gbps のネットワークトラフィックを監視できる．

また，図 8 から分かるように，実験中の全期間平均処理遅延はおよそ 357 msec であった．最長の場合ではおよそ 777 msec の処理遅延が生じたセッションが存在するものの，全期間の 80 パーセンタイルはおよそ 430 msec であり，

表 7 Small の送信結果
Table 7 Transmisson result of Small.

	Small
送信パケット数	39,321,600
送信量（GB）	1.98
送信時間（sec）	566.23
平均送信パケット数（pps）	69,444.44
平均送信量（Mbps）	30.00

大半のセッションでは 500 msec 以内で処理が完了した．

なお，本フレームワークの性質上，クラスタ構成や各種パラメータのチューニングによってシステム全体の処理遅延が変化する．このため，本論文においては処理遅延の妥当性については議論しない．

また，Capture 2 の測定結果については Capture 1 と同様の傾向であったため割愛する．

5.3 高負荷を生じるトラフィックを利用した性能評価

UNSW-NB15 data set を利用した 5.2 節の実験では，セッションあたりのパケットサイズの合計が平均数十 kByte 程度であったため，1 Gbps のネットワークトラフィックを処理しても 2,500 session/sec 程度の負荷であった．しかしながら，実際にはセッションあたりのパケット数が極端に少なく，加えてそれぞれのパケットサイズも小さくなる場合がある．このような場合には，パケット数に対してセッション数が十分に少なくならないため，システムに対して高い負荷が生じる恐れがある．このため，意図的に高負荷を生じるようなネットワークトラフィックを発生させてシステムの性能を評価した．

意図的に高負荷を生じるために，セッションが 2 パケットで構成される通信を大量に記録したパケットキャプチャファイル（Small）を用意した．通信の内容は，ホスト A から TCP の SYN パケットをホスト B に送信したものの，ホスト B は RST パケットをホスト A へ送信するというものである．このような通信は，Bro IDS の解析によって 1 つのセッションとして扱われる（接続拒否）．Small には，ホスト A を 192.168.0.0/16 の全ホスト，ホスト B を 10.0.0.1 としたときの全パケットが含まれる．

Tcpreplay を利用して送信量を変化させながら Small を送信したところ，30 Mbps の時点で Elasticsearch においてストレージ速度がボトルネックになったため，以降では 30 Mbps 送信時の結果を示す．Small を Tcpreplay により 300 回繰り返し送信した結果を**表 7** に示す．表 7 に示したように，Small を 30 Mbps でおよそ 570 秒間送信し続けた．平均送信パケット数はおよそ 6.9 万 pps であったため，システムでは 1 秒間におよそ 3.5 万のセッションを処理できなければならない．

図 9，**図 10**，**図 11** に，それぞれ Small 送信時のスルー

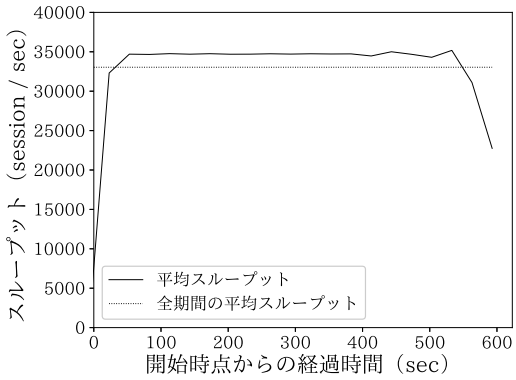

図 9 実験中のスループットの推移（Small）
Fig. 9 Transition of throughput during experiment (Small).

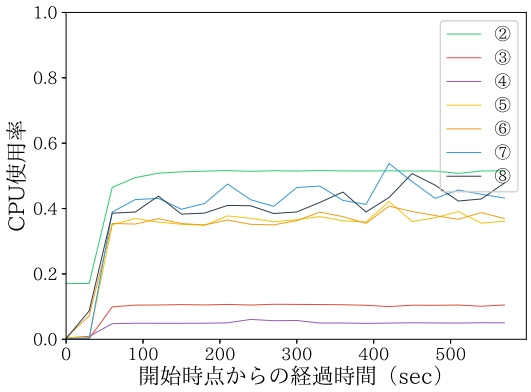

図 10 実験中の CPU 使用率の推移（Small）
Fig. 10 Transition of CPU usage during experiment (Small).

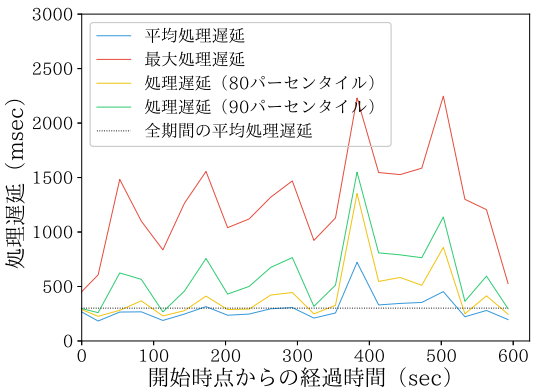

図 11 実験中の処理遅延の推移（Small）
Fig. 11 Transition of lag during experiment (Small).

プット，CPU 使用率，処理遅延を示す．図 9 から分かるように，1 秒間におよそ 3.5 万セッションを処理できた．しかしながら，図 10 から分かるように，多くのマシンで CPU 使用率が 40〜50%程度まで上昇した．CPU 使用率が大きく上昇したのは，Bro Cluster を実行しているマシン（②）と，Logstash および Elasticsearch を実行しているマシン（⑤，⑥，⑦，⑧）であった．一方で，Kafka Broker および Kafka Streams アプリケーションを実行しているマシン（③，④）では CPU 使用率が 10%未満であった．

また，図 11 から分かるように，実験中の全期間平均処理遅延はおよそ 300 msec であった．一部で処理遅延が大幅に増加した区間が存在するものの，全期間の 80 パーセンタイルはおよそ 366 msec であり，大半のセッションの処理遅延は平均と近い値になった．一方で，最大の場合では 2,247 msec の処理遅延が発生したり，全期間の 90 パーセンタイルが 683 msec となったりするなど，低負荷時と比べて高負荷時には処理遅延が大きくなるセッションが増える傾向があった．

6. フレームワークの性能限界についての考察

本フレームワークの各機能は分散処理フレームワークを利用して実装してあるため，基本的にスケーラブルである．たとえば，Bro Cluster については 5 ノードのクラスタにおいてピーク時でおよそ 24 Gbps のネットワークトラフィックを処理した例がある [33]．また，Apache Kafka については 3 ノードのクラスタにおいて秒間およそ 200 万メッセージの書き込みが行えた例がある [34]．さらに，Elasticsearch についても 15 ノードのクラスタにおいて秒間およそ 100 万ドキュメントの書き込みが行えた例がある [35]．

しかしながら，一部の機能については処理の分配に制限があるために，悪条件下ではスケーラビリティが失われる場合がある．以降では，スケーリングの妨げになる恐れのある機能について詳細に述べる．

6.1 Bro Cluster によるセッション構築の限界

Bro Cluster によるセッション構築時には，マシン間およびプロセス間でのパケットの分配によってスケーラビリティを確保している．セッションを正しく構築するためには，セッションを構築するすべてのパケットが同じ Bro IDS プロセスで処理されるように分配しなければならない．このため，Bro Cluster では標準で 4 つ組（送信元/宛先 IP アドレス・送信元/宛先ポート番号）を利用してパケットを分配する．たとえば，ホスト A の C ポートからホスト B の D ポートへのセッションの場合は，A:C から B:D へのパケットと B:D から A:C へのパケットを同じ Bro IDS プロセスに分配する．しかしながら，このようにパケットを分配する場合，最悪の場合にはすべてのセッション構築が単一の Bro IDS プロセスに割り当てられる恐れがある．

単一の Bro IDS プロセスのスループットは，処理対象のネットワークトラフィックによって変化する．実際に，5.2 節の実験では 1 プロセスあたりおよそ 36,810 pps を処理しても CPU 使用率が 30%程度であったのに対し，5.3 節の実験では 1 プロセスあたりおよそ 11,574 pps を処理して CPU 使用率が 50%程度であった．5.3 節の実験で明らかになったように，Bro IDS のスループットはセッションの開始時と終了時に大きく低下する．よって，5.3 節で示したようなセッションが単一の Bro IDS プロセスに割り当てられる場合にスループットの面でおよそ最悪の性能を

招く．表 4 ②のマシンを利用して実験したところ，およそ 10 Mbps が限界であった．したがって，最悪の場合にはシステム全体でおよそ 10 Mbps のネットワークトラフィックまでしか監視できない．

5.3 節で示したようなセッションは，たとえば SYN flood のような DoS 攻撃の発生時に数多く現れることが予想されるため，本フレームワークにおいても考慮しなければならない．しかしながら，セッション構築部分を Bro Cluster に依存する現在の設計では対処できず，最悪の場合 Bro Cluster 内の Bro IDS プロセスがクラッシュする恐れがある．対策としては，SYN flood のような DoS 攻撃をフィルタして Bro Cluster に入力しない仕組みを用意することや，本フレームワークに必要な機能に絞ってより軽量化したセッション構築のための仕組みを用意することがあげられる．

6.2 ホストベース特徴量の抽出部分の限界

表 1 に示したホストベース特徴量を抽出するために，基本特徴量を Kafka Broker に保存する際にはセッションの宛先アドレスを Kafka メッセージのキーに設定する．これにより，Kafka Streams を利用したホストベース特徴量の抽出アプリケーションにおいて，宛先アドレスごとにホストベース特徴量を抽出するスレッドが固定される．しかしながら，このように基本特徴量を分配する場合，最悪の場合にはすべての基本特徴量が単一の特徴抽出スレッドに割り当てられる恐れがある．

単一の特徴抽出スレッドのスループットを測定するために，Kafka Broker に対して大量のダミーデータを投入して特徴抽出に要した時間を測定した．投入したダミーデータは，5.3 節で示したような 1 セッションが 2 パケットで構成される通信である．ただし，送信元アドレスを 192.168.0.1，宛先アドレスを 10.0.0.0/22 の全ホストとして，宛先アドレスごとに 5 万セッションが構築されるようにした．さらに，宛先アドレスごとに連続して 5 万セッション送信する場合（以下パターン A）と，宛先アドレスを毎回ずらしながら 5 万セッション送信する場合（以下パターン B）の 2 通りで測定した．パターン A では，同じ宛先アドレスのセッションが連続するため，ホストベース特徴量の抽出時に CPU キャッシュによる高速化が期待できる．一方で，パターン B では毎回異なる宛先アドレスに対して処理するため，CPU キャッシュによる高速化が期待できない過酷な条件である．表 4 ④のマシンを利用して実験したところ，パターン A ではおよそ 24.38 万 session/sec，パターン B ではおよそ 10.67 万 session/sec が限界であった．予想のとおり，CPU キャッシュによる高速化が期待できるパターン A の場合と比較して，パターン B の場合ではスループットが低下した．

5.2 節では，1 Gbps のネットワークトラフィックを処理してもピーク時でおよそ 2,500 session/sec であった．パターン A の場合，単純計算で 100 Gbps 時のピーク流量をおよそ 25 万 session/sec としても，単一のスレッドでほぼすべてを特徴抽出できる．パターン B の場合，単純計算で 40 Gbps 時のピーク流量をおよそ 10 万 session/sec としても，単一のスレッドで十分に特徴抽出できる．

また，5.3 節のように 2 パケットで 1 つのセッションを構築する場合について，セッションあたりのパケットサイズの合計を 128 Byte と仮定する．このとき，パターン A が 1 秒あたり 24.38 万セッション，パターン B が 1 秒あたり 10.67 万セッションを処理できると仮定すれば，理論上はそれぞれ 238.09 Mbps，104.20 Mbps のネットワークトラフィックを処理できる．

基本的には特徴抽出スレッドの性能は十分であるものの，いくつかの場合に単一の特徴抽出スレッドに処理が集中する恐れがある．

まず，複数の宛先アドレスの割当てがたまたま単一の特徴抽出スレッドに偏ってしまう場合がある．このような場合には，基本特徴量用の Topic の Partition 数を増やし，特徴抽出スレッドの数を増やすことで偏りを緩和できる可能性がある．

次に，6.1 節と同様に，DoS 攻撃のような通信によって単一の宛先アドレスに向けて大量のセッションが生じる場合がある．このような場合には，特徴抽出スレッドの数にかかわらず単一の特徴抽出スレッドに割当てが集中するため本フレームワークでは対処できない．対策としては，6.1 節と同様に DoS 攻撃のような通信をフィルタすることがあげられる．あるいは，宛先アドレスごとの集約が必要な現在のホストベース特徴量を見直し，分配のしやすさを考慮して新たな特徴量を設計することで解決できる可能性がある．

なお，6.1 節の Bro Cluster の場合とは異なり，単一の特徴抽出スレッドに対して限界を超えるバーストが発生したとしても，Kafka Broker が吸収できるほど短時間であれば処理遅延は発生するもののすべてのセッションを正常に処理できる．

7. セッションベース特徴量の差異による分類性能への影響評価

本フレームワークにおいては，機械学習ベースの NIDS についての既存研究をそのまま活用するために，KDD Cup 1999 Data 形式のセッションベースの特徴量を利用できるようにした．しかしながら，3.1 節で述べたように，KDD Cup 1999 Data 形式のセッションベース特徴量の一部を除外したことにより，KDD Cup 1999 Data や KDD Cup 1999 Data の改良版である NSL-KDD dataset を利用して評価された既存研究の分類機構を本フレームワークに組み込む際に，利用できるセッションベース特徴量の差異に

表 8 KDD Cup 1999 Data の分類結果
Table 8 Classification results of KDD Cup 1999 Data.

	正解率（%）	再現率（%）	適合率（%）
（A）	93.06	91.54	99.83
（B）	92.86	91.25	99.87
（C）	92.75	91.11	99.88

よって分類性能が低下する恐れがある．このため，KDD Cup 1999 Data に対して機械学習による分類を適用してセッションベース特徴量の差異による分類性能への影響を評価した．

学習用データには，全学習用データのうち 1 割のデータを含む “kddcup.data_10_percent” を利用し，評価用データには，評価用データとして用意されている “corrected” を利用した．利用した特徴量の集合は，（A）KDD Cup 1999 Data と本フレームワークに共通して含まれる 11 種類の特徴量を利用した場合と，（B）Kyoto Dataset に含まれる 4 種類の time-based traffic features に（A）の 11 種類の特徴量を加えた 15 種類の特徴量を利用した場合，（C）KDD Cup 1999 Data の全特徴量である 41 種類の特徴量を利用した場合の 3 通りである．

表 8 に，3 通りの特徴量の集合に対してそれぞれランダムフォレストによる分類を適用した結果を示す．

まず，表 8 の（B）と（C）に着目すると分類性能の変化はわずかである．この結果から，KDD Cup 1999 Data の全特徴量のうち，Kyoto Dataset に含まれる特徴量が含まれていれば分類性能への大きな影響は生じないと考えられる．

また，表 8 の（A）と（B）に着目しても分類性能の変化はわずかである．この結果から，本フレームワークにおいて time-based traffic features を除外したことによる分類性能への影響も軽微であると考えられる．

したがって，KDD Cup 1999 Data や NSL-KDD dataset を前提とした既存研究の分類機構を本フレームワークに組み込んでも分類性能への影響は軽微であり，本フレームワークで利用できる特徴量のみであっても既存研究の分類機構を活用できると考えられる．

よって，本フレームワークにおいては KDD Cup 1999 Data や NSL-KDD dataset，Kyoto Dataset を前提とした既存研究の分類機構をそのまま利用できる．加えて，本フレームワークが想定する KDD Cup 1999 Data 系のセッションベース特徴量とは異なるため分類性能の低下を招く恐れはあるものの，利用者側で分類性能について検証できる場合には，UNSW-NB15 data set やセッションベース特徴量を採用したその他の NIDS 評価用データセットを前提とした一部の既存研究の分類機構も利用できる．

8. まとめ

本論文では，セッションベース特徴量を利用した，スケーラブルな機械学習ベースの NIDS を構築するための分散処理フレームワークを提案した．ストリーム処理が可能な既存の分散処理フレームワークを利用して各機能を実装することで，通信の発生後即座に検知結果を得られるだけでなくスケーラビリティも両立させた．

提案したフレームワークを利用して実際に小規模なクラスタ上にシステムを構築し，システム全体のスループットや処理遅延，各マシンの CPU 使用率といった指標で性能を評価した．

現実的なネットワークトラフィックとして，UNSW-NB15 data set に含まれる PCAP ファイルを利用した．1 Gbps のネットワークトラフィックはおよそ 2,500 session/sec のストリームデータとなったものの，一般消費者向けの安価なマシンで構成した小規模なクラスタであっても余裕を持って処理できた．さらに，実験中の平均処理遅延はおよそ 357 msec であり，処理遅延の 80 パーセンタイルもおよそ 430 msec であったため，大半のセッションでは 500 msec 以内の処理遅延であった．この結果は，安価なマシンで構築した小規模なクラスタであっても 1 Gbps のネットワークを 500 msec 程度の処理遅延で監視できることを示したものであり，より実用的な機械学習ベースの NIDS を構築するための足がかりとなるものである．

一方，高負荷を生じるネットワークトラフィックとして，セッションを構成するパケットが極端に少なくなる状況での評価では，わずか 30 Mbps のネットワークトラフィックであってもクラスタ全体が高負荷状態に陥ることを確認した．

また，提案したフレームワークの設計上の性能限界についても考察した．各機能における処理の分配で偏りが生じない理想的な状況下においては，現在の設計であっても十分なスケーラビリティを確保できる．しかしながら，いくつかの場合では理想的に処理を分配できず，システム全体のスループットが現在のネットワーク環境に対応できないことが明らかになった．

より大きなクラスタ上でのスケーラビリティについての評価と，理想的に処理を分配できない場合の最悪性能の向上が今後の課題である．加えて，本フレームワークを公開するための準備も進める予定である．

謝辞 本研究の一部は，JSPS 科研費 17K00076，16K00071 の支援により行った．

参考文献

[1] Sumits, A.: The History and Future of Internet Traffic, Cisco (online), available from ⟨https://blogs.cisco.com/sp/the-history-and-future-of-internet-traffic⟩ (accessed 2018-10-30).
[2] Cisco: Cisco Visual Networking Index: Forecast and Methodology, 2016–2021, Cisco (online), available from ⟨https://www.cisco.com/c/en/us/solutions/collateral/

service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html⟩ (accessed 2018-10-30).
[3] Chandola, V., Banerjee, A. and Kumar, V.: Anomaly Detection: A Survey, *ACM Comput. Surv.*, Vol.41, No.3, pp.15:1–15:58 (online), DOI: 10.1145/1541880.1541882 (2009).
[4] Buczak, A.L. and Guven, E.: A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection, *IEEE Communications Surveys Tutorials*, Vol.18, No.2, pp.1153–1176 (online), DOI: 10.1109/COMST.2015.2494502 (2016).
[5] Kato, K. and Klyuev, V.: Development of a network intrusion detection system using Apache Hadoop and Spark, *2017 IEEE Conference on Dependable and Secure Computing*, pp.416–423 (online), DOI: 10.1109/DESEC.2017.8073860 (2017).
[6] Manzoor, M.A. and Morgan, Y.: Real-time Support Vector Machine based Network Intrusion Detection system using Apache Storm, *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference* (*IEMCON*), pp.1–5 (online), DOI: 10.1109/IEMCON.2016.7746264 (2016).
[7] Dahiya, P. and Srivastava, D.K.: Network Intrusion Detection in Big Dataset Using Spark, *Procedia Computer Science*, Vol.132, pp.253–262 (online), DOI: https://doi.org/10.1016/j.procs.2018.05.169 (2018).
[8] Rathore, M.M., Ahmad, A. and Paul, A.: Real Time Intrusion Detection System for Ultra-high-speed Big Data Environments, *J. Supercomput.*, Vol.72, No.9, pp.3489–3510 (online), DOI: 10.1007/s11227-015-1615-5 (2016).
[9] Dromard, J., Roudière, G. and Owezarski, P.: Unsupervised Network Anomaly Detection in Real-Time on Big Data, *New Trends in Databases and Information Systems*, Morzy, T., Valduriez, P. and Bellatreche, L. (Eds.), Cham, Springer International Publishing, pp.197–206 (2015).
[10] Marchal, S., Jiang, X., State, R. and Engel, T.: A Big Data Architecture for Large Scale Security Monitoring, *2014 IEEE International Congress on Big Data*, pp.56–63 (online), DOI: 10.1109/BigData.Congress.2014.18 (2014).
[11] Gao, Y., Liu, Y., Jin, Y., Chen, J. and Wu, H.: A Novel Semi-Supervised Learning Approach for Network Intrusion Detection on Cloud-Based Robotic System, *IEEE Access*, Vol.6, pp.50927–50938 (online), DOI: 10.1109/ACCESS.2018.2868171 (2018).
[12] Al-Qatf, M., Lasheng, Y., Al-Habib, M. and Al-Sabahi, K.: Deep Learning Approach Combining Sparse Autoencoder With SVM for Network Intrusion Detection, *IEEE Access*, Vol.6, pp.52843–52856 (online), DOI: 10.1109/ACCESS.2018.2869577 (2018).
[13] UCI KDD Archive: KDD Cup 1999 Data (online), available from ⟨http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html⟩ (accessed 2018-10-30).
[14] University of New Brunswick: NSL-KDD dataset (online), available from ⟨https://www.unb.ca/cic/datasets/nsl.html⟩ (accessed 2018-10-30).
[15] Song, J., Takakura, H., Okabe, Y., Eto, M., Inoue, D. and Nakao, K.: Statistical Analysis of Honeypot Data and Building of Kyoto 2006+ Dataset for NIDS Evaluation, *Proc. 1st Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, *BADGERS '11*, New York, NY, USA, ACM, pp.29–36 (online), DOI: 10.1145/1978672.1978676 (2011).
[16] 多田竜之介，小林良太郎，嶋田 創，高倉弘喜：NIDS 評価用データセット：Kyoto 2016 Dataset の作成，情報処理学会論文誌，Vol.58, No.9, pp.1450–1463 (2017).
[17] Takakura, H.: Traffic Data from Kyoto University's Honeypots, Takakura, H. (online), available from ⟨http://www.takakura.com/Kyoto_data/⟩ (accessed 2018-10-30).
[18] Moustafa, N. and Slay, J.: UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set), *2015 Military Communications and Information Systems Conference* (*MilCIS*), pp.1–6 (online), DOI: 10.1109/MilCIS.2015.7348942 (2015).
[19] Moustafa, N. and Slay, J.: The Evaluation of Network Anomaly Detection Systems: Statistical Analysis of the UNSW-NB15 Data Set and the Comparison with the KDD99 Data Set, *Inf. Sec. J.: A Global Perspective*, Vol.25, No.1-3, pp.18–31 (online), DOI: 10.1080/19393555.2015.1125974 (2016).
[20] The Apache Software Foundation: Apache Kafka, The Apache Software Foundation (online), available from ⟨https://kafka.apache.org/⟩ (accessed 2018-11-03).
[21] The Bro Project: The Bro Network Security Monitor (online), available from ⟨https://www.bro.org/index.html⟩ (accessed 2018-11-03).
[22] The Apache Software Foundation: Apache Avro (online), available from ⟨https://avro.apache.org/⟩ (accessed 2018-11-03).
[23] The Apache Software Foundation: Apache Spark (online), available from ⟨https://spark.apache.org/⟩ (accessed 2018-11-03).
[24] The Apache Software Foundation: Apache Storm (online), available from ⟨http://storm.apache.org/⟩ (accessed 2018-11-03).
[25] The Apache Software Foundation: Apache Samza (online), available from ⟨http://samza.apache.org/⟩ (accessed 2018-11-03).
[26] The Apache Software Foundation: Apache Flink (online), available from ⟨https://flink.apache.org/⟩ (accessed 2018-11-03).
[27] Edenhill, M.: librdkafka – The Apache Kafka C/C++ client library, Edenhill services (online), available from ⟨https://github.com/edenhill/librdkafka⟩ (accessed 2018-11-03).
[28] Elastic: Elasticsearch (online), available from ⟨https://www.elastic.co/jp/products/elasticsearch⟩ (accessed 2018-11-03).
[29] Elastic: Kibana (online), available from ⟨https://www.elastic.co/jp/products/kibana⟩ (accessed 2018-11-03).
[30] Elastic: Logstash (online), available from ⟨https://www.elastic.co/jp/products/logstash⟩(accessed 2018-11-04).
[31] AppNeta: Tcpreplay (online), available from ⟨http://tcpreplay.appneta.com/⟩ (accessed 2018-11-04).
[32] Wireshark developers: Wireshark (online), available from ⟨https://www.wireshark.org/⟩ (accessed 2018-11-04).
[33] Berkeley Lab: 100G Intrusion Detection (online), available from ⟨http://go.lbl.gov/100g⟩ (accessed 2018-11-21).
[34] Kreps, J.: Benchmarking Apache Kafka: 2 Million Writes Per Second (On Three Cheap Machines), LinkedIn (online), available from ⟨https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines⟩ (accessed 2018-11-21).
[35] Kothari, S.: Benchmarking Elasticsearch: 1 Million

Writes per Sec, appbase.io (online), available from ⟨https://medium.appbase.io/benchmarking-elasticsearch-1-million-writes-per-sec-bf37e7ca8a4c⟩ (accessed 2018-11-21).

付　　録

A.1 独自形式によるデータ交換の効果

本フレームワークにおいては，システム性能の向上のために中間データ形式として表 2 のような独自形式を採用した．以降では，独自形式の採用による効果をデータサイズおよびシリアライズ時とデシリアライズ時の処理時間の観点からそれぞれ述べる．

A.1.1 独自形式の採用によるデータサイズの改善

JSON 形式と独自形式のそれぞれについて，データサイズの最大値を**表 A·1** に示す．

表 A·1 に示したように，JSON 形式の場合，最大時には基本特徴量のみの場合で 449 Byte，完全な特徴量の場合で 590 Byte の領域を必要とする場合がある．このうち，属性名を表すキーや文字列を表すダブルクォーテーションのような余分な領域は，基本特徴量のみの場合で 238 Byte，完全な特徴量の場合で 364 Byte である．したがって，本フレームワークにおいて JSON 形式を利用する場合には，全体の半分を超える領域が本質的に意味を持たない余分な領域であり，データサイズの面で非効率である．

一方で，独自形式の場合，最大時でも基本特徴量のみの場合で 143 Byte，完全な特徴量の場合で 153 Byte の領域しか必要としない．JSON 形式の場合と比較しておよそ 4 分の 1 から 3 分の 1 程度の領域であり，より効率的である．

A.1.2 独自形式の採用によるシリアライズ時およびデシリアライズ時の処理時間の改善

シリアライズ時およびデシリアライズ時の処理時間を測定するために，あらかじめ Bro IDS でセッション構築した際のログファイルを利用し，セッションログファイルからシリアライズとデシリアライズを行った場合の処理時間を測定するプログラムを作成した．測定プログラムでは，シリアライズ時およびデシリアライズ時に各過程で必要とする形式の入力データをあらかじめメモリ上に読み込んでおき，メモリ上の入力データからのシリアライズおよびデシリアライズの処理時間を計測できるようにした．このため，セッションログファイルからダミーデータを作成する時間を除外して処理時間を測定できる．

実験には CPU に 3.0 GHz で動作する AMD Ryzen 7 1700 を搭載したマシンを利用し，UNSW-NB15 data set の PCAP ファイルのうちおよそ 1 GB 分を解析したセッションログファイルを入力した．図 2 の② → ① 間のシリアライズを（A），① → ③ 間のデシリアライズを（B），③ → ① 間のシリアライズを（C），① → ④ 間のデシリアライズを（D）として，JSON 形式と独自形式のそれぞれについて，各過程におけるデータ 1 件あたりのシリアライズおよびデシリアライズの処理時間を**表 A·2** に示す．

まず，表 A·2（A）の部分は，Bro IDS に組み込んだ独自プラグイン（C++で記述）上で基本特徴量をシリアライズした場合の JSON 形式と独自形式の比較である．（A）では，セッション情報のうち基本特徴量に該当するすべての情報をシリアライズしなければならない．独自形式を利用した場合には，JSON 形式と比較して 10 倍程度高速である．

次に，表 A·2（B）の部分は，ホストベース特徴量の抽出アプリケーション（Java で記述）上で基本特徴量をデシリアライズした場合の JSON 形式と独自形式の比較である．（B）では，ホストベース特徴量を抽出するためには，基本特徴量のすべての情報を読み出す必要はなく，送信元・宛先アドレスやサービス名などの一部の情報のみを読み出せばよい．しかしながら，JSON 形式では一部の情報のみが必要な場合であってもすべての情報を読み出さなければならないため，無駄な処理が生じてしまう．一方で，独自形式ではバイナリデータのどの部分に必要な情報で記録されているかを特定できるため，必要な情報のみを無駄なく読み出せる．この結果，独自形式を利用した場合には JSON 形式と比較して 50 倍程度高速である．

そして，表 A·2（C）の部分は，ホストベース特徴量の抽出アプリケーション（Java で記述）上で完全な特徴量をシリアライズした場合の JSON 形式と独自形式の比較である．（C）では，JSON 形式の場合，抽出したホストベース特徴量を基本特徴量に加えた完全な特徴量の JSON デー

表 A·1 JSON 形式と独自形式のデータサイズについての比較
Table A·1 Comparison between JSON format and the original data format for data size.

	IPv4（最大）	IPv6（最大）
JSON 形式（基本特徴量）	401 Byte	449 Byte
独自形式（基本特徴量）	119 Byte	143 Byte
JSON 形式（完全な特徴量）	542 Byte	590 Byte
独自形式（完全な特徴量）	129 Byte	153 Byte

表 A·2 JSON 形式と独自形式の処理時間についての比較
Table A·2 Comparison between JSON format and the original data format for processing time.

	処理時間（μs）		1 秒あたりの処理件数（件）	
	JSON 形式	独自形式	JSON 形式	独自形式
（A）	2.6	0.25	384,607	3,976,983
（B）	3.1	0.061	324,843	16,487,296
（C）	2.5	0.071	394,300	13,988,659
（D）	3.2	0.34	313,559	2,912,226

タを得るためには，すべての情報をシリアライズし直さなければならない．一方で，独自形式の場合，抽出したホストベース特徴量を基本特徴量に結合するだけで完全な特徴量が得られる．この結果，独自形式を利用した場合にはJSON 形式と比較して 35 倍程度高速である．

最後に，表 A·2（D）の部分は，完全な特徴量の形式変換アプリケーション（Java で記述）上で完全な特徴量をデシリアライズした場合の JSON 形式と独自形式の比較である．（D）では，データの形式変換のためには完全な特徴量のすべての情報を読み出さなければならない．独自形式を利用した場合には，JSON 形式と比較して 9 倍程度高速である．

このように，独自形式を採用した場合には，シリアライズ時およびデシリアライズ時の処理時間が JSON 形式と比較して少なくとも 10 分の 1 程度であり，各処理過程に必要なマシンの数を抑制できる．特に，6.2 節で述べたようにホストベース特徴量の抽出処理部分ではスレッドあたりのスループットがシステムの最悪性能を決めるため，表 A·2 の（B）および（C）部分の処理時間の削減はシステムの最悪性能の向上に大きく貢献するものである．

多田 竜之介

2017 年豊橋技術科学大学工学部情報・知能工学課程卒業．同年同大学大学院工学研究科情報・知能工学専攻博士前期課程入学．ネットワークセキュリティの研究に従事．

中村 純哉

2006 年豊橋技術科学大学工学部知識情報工学課程卒業．2008 年同大学大学院工学研究科知識情報工学専攻修了．2014 年大阪大学大学院情報科学研究科コンピュータサイエンス専攻博士後期課程修了．博士（情報科学）．同年豊橋技術科学大学情報メディア基盤センター特任助教．2017 年豊橋技術科学大学情報メディア基盤センター助教．分散アルゴリズム，分散システム，情報システムの研究に従事．特に，システムの耐故障性に興味を持つ．電子情報通信学会会員．

大村 廉 （正会員）

1999 年慶應義塾大学理工学部電気工学科卒業．2001 年同大学大学院理工学研究科計算機科学専攻前期博士課程修了．2004 年同大学院理工学研究科開放環境科学専攻後期博士課程修了．博士（工学）．同年（株）国際電気通信基礎技術研究所研究員．2007 年慶應義塾大学理工学部情報工学科助教．2010 年豊橋技術科学大学情報・知能工学系講師．2019 年より豊橋技術科学大学情報・知能工学系准教授．電気情報通信学会，IEEE，ACM 各会員．ユビキタスコンピューティング，ウェアラブルコンピューティング，コンテキストアウェアシステム，センサネットワーク，行動認識技術等の研究に従事．

小林 良太郎 （正会員）

1995 年名古屋大学工学部電子情報学科卒業．1997 年同大学大学院工学研究科電子情報学専攻博士課程前期課程修了．2000 年同大学院工学研究科電子情報学専攻博士課程後期課程満了．工学博士．2000 年名古屋大学大学院工学研究科電子情報学専攻助手．2008 年豊橋技術科学大学工学部講師．2016 年豊橋技術科学大学大学院工学研究科准教授．2017 年工学院大学情報学部准教授．2019 年工学院大学情報学部教授．1999 年情報処理学会山下記念研究賞受賞．2002 年情報処理学会論文賞受賞．サイバーセキュリティ，計算機アーキテクチャの研究に従事．