DDoS攻撃検知にかかる処理時間の短縮に関する提案

喜村 柊太[1,a)]　小林 孝史[2,b)]

概要：既存の攻撃検知手法では，パケットキャプチャを行い，特徴量を抽出し，攻撃判定を行うまでの合計処理時間が長くかかる課題がある．合計処理時間はパケットキャプチャ時間（秒数）に依存しており，可能な限り短い時間で行うことが望ましい．本研究では，OpenStack ベースのクラウド環境に対する DDoS 攻撃を対象とする．エントロピーと機械学習を用いた既存の攻撃検知システムについて，検知性能に影響を与えないような最短のパケットキャプチャ時間を調査し，提案する．実験の結果，より短いキャプチャ時間であっても攻撃検知性能を維持し，攻撃検知までの処理時間を短縮できることが確認された．

キーワード：DDoS 攻撃検知，エントロピー，機械学習，OpenStack，クラウドコンピューティング

Proposal for Reducing the Processing Time for DDoS Attack Detection

Shuta Kimura[1,a)]　Takashi Kobayashi[2,b)]

Abstract: Existing DDoS attack-detection approaches suffer from prolonged total processing time, including packet capture, feature extraction, and attack classification. The overall processing time depends on the packet-capture-window duration; minimizing this duration is desirable. In this study, we focus on detecting DDoS attacks in OpenStack-based cloud environments. For detection systems employing entropy measures and machine learning, we investigate and determine the minimum packet-capture duration that does not degrade detection performance. Experimental results demonstrate that detection performance can be maintained even with shorter capture windows, thereby reducing the total detection latency.

Keywords: DDoS attack detection, entropy, machine learning, OpenStack, cloud computing

1. はじめに

DDoS 攻撃（Distributed Denial of Service attack）は，サイバーセキュリティにおいてシステムの可用性を脅かす重大な脅威である．インターネットイニシアティブ（IIJ）[1] によると，2024 年に観測された最大規模の攻撃は 174.80 Gbps に達し，最長継続時間は約 3 時間 24 分に及んだ．このように大規模かつ長時間にわたる攻撃が現実に確認されており，有効な対策が求められている．特に，OpenStack をはじめとするクラウド環境に対する攻撃は，共有リソースを消耗させ，全体をサービス不能に陥らせる危険性があるため，迅速な検知と緩和が不可欠である．

既存の攻撃検知手法では，パケットを一定時間キャプチャしてフロー統計量を生成し，機械学習モデルで判定している．しかしキャプチャ時間が長いほど，特徴量抽出・攻撃判定までの合計処理時間が増大し，リアルタイム性が損なわれる．本研究が対象とする既存システムでは，エントロピーモジュールが最短で 3 秒周期でトリガーを発し，機械学習を用いた検知モジュールがパケットキャプチャと攻撃判定を行う．このとき，検知モジュールにおける合計処理時間を 3 秒未満に収められれば，モジュール間がより連携して動作し，かつ両モジュールでかかる処理時間を合計した全体の検知時間の短縮が期待できる．

そこで本研究の目的は，検知性能を維持したまま攻撃を検知できる最短のパケットキャプチャ時間を明らかにし，

[1] 関西大学大学院 総合情報学研究科 知識情報学専攻
Intelligent Informatics Major, Graduate School of Informatics, Kansai University
[2] 関西大学 総合情報学部
Faculty of Informatics, Kansai University
[a)] k149017@kansai-u.ac.jp
[b)] taka-k@kansai-u.ac.jp

攻撃検知までの合計処理時間を短縮することである．対象は OpenStack ベースのクラウド環境に対する DDoS 攻撃（本研究では SYN Flood）とする．

本研究の主な貢献は以下のとおりである．

- **キャプチャ時間―性能トレードオフの体系的評価**：OpenStack 環境で収集したデータに対し，ロジスティック回帰（LR），ナイーブベイズ（NB），ランダムフォレスト（RF）の 3 モデルについて，キャプチャ時間を段階的に変化させ，検知成功率・処理時間・抽出フロー数を比較できる評価設定を提示した（各条件 10 回試行で評価）．
- **最短キャプチャ時間の特定**：検知成功の定義（各条件 10/10 回で検知成立）を満たしつつ，ロジスティック回帰で 0.307 秒，ナイーブベイズで 0.705 秒，ランダムフォレストで 0.237 秒までキャプチャ時間を短縮できることを示した．
- **合計処理時間の顕著な短縮**：最良構成であるランダムフォレストにおいて，キャプチャ 0.237 秒時の検知モジュールの合計処理時間は 1.83 秒であり，従来の 3.000 秒キャプチャ時の 6.999 秒と比べて 5.169 秒短縮できることを実証した．この結果は，検知モジュールの処理が 3 秒周期トリガー内に収まり，モジュール間連携の改善と検知のリアルタイム性向上に資することを示唆する．

2. エントロピー（平均情報量）

2.1 概要

情報理論におけるエントロピーとは，あるデータ集合における不確実性や情報のばらつきを表す指標である．本研究では，ネットワークトラフィック中の宛先 IP アドレスのエントロピーを次の式で計算する：$H = -\sum_{i=1}^{N} p_i \log_2(p_i)$．ここで，$p_i$ は宛先 IP アドレス i の出現確率を示し，N は観測された全宛先 IP アドレスの数である．エントロピーが低い場合は，トラフィックが特定の IP アドレスに集中していることを示し，DDoS 攻撃の兆候となる可能性がある．

2.2 計算例

エントロピーの概念を具体的に理解するために，簡単な計算例を示す．表 1 は，あるネットワークトラフィックにおける宛先 IP アドレスの出現頻度を示している．

表 1 宛先 IP アドレスの出現頻度のエントロピー計算例

宛先 IP アドレス	出現回数
192.168.1.1	50
192.168.1.2	30
192.168.1.3	20
合計	100

このデータに基づき，各 IP アドレスの出現確率 p_i を計算すると，

$$p_1 = \frac{50}{100} = 0.5, \quad p_2 = \frac{30}{100} = 0.3, \quad p_3 = \frac{20}{100} = 0.2$$

となる．これをエントロピーの式に代入すると，

$$H = -\sum_{i=1}^{3} p_i \log_2(p_i)$$

$$= -(0.5 \log_2 0.5 + 0.3 \log_2 0.3 + 0.2 \log_2 0.2)$$

$$\approx -(0.5 \times (-1) + 0.3 \times (-1.737) + 0.2 \times (-2.322))$$

$$\approx 0.5 + 0.5211 + 0.4644 \approx 1.4855$$

したがって，この例における宛先 IP アドレスのエントロピーは約 1.49 となる．この値が高いほど，トラフィックがさまざまな IP アドレスに分散していることを示し，逆に低い場合は特定の IP アドレスに集中していることを意味する．特に DDoS 攻撃では，攻撃対象の IP アドレスにパケットが集中するため，エントロピーが低下する傾向にある．

このように，エントロピーを算出することでネットワークトラフィックの特性を定量的に評価し，異常検知の指標として活用することが可能である．

3. 関連研究

Patil らの研究 [2] では，Controller，Neutron，Compute，Compute2 の 4 ノードからなる OpenStack 環境が用いられた．外部ネットワークから Compute ノード上で動作するインスタンスに対する攻撃が検知対象である．外部ネットワークから送信されたパケットは，外部ネットワークから Neutron ノードを経由し，Compute ノード上のインスタンスに到達するという経路で行われる．攻撃検知手法として，エントロピーと機械学習を用いた手法が提案されている．

エントロピーモジュールは Neutron ノードにおいて動作する．3 秒間パケットキャプチャを行い，クラウド環境の外部から内部に向けて届くパケットについて，宛先 IP アドレスの出現数から情報エントロピーを計算するという一連の動作を繰り返し行う．このとき，エントロピーがしきい値である 1.0 を下回れば，Compute ノードで動作する検知モジュールにトリガー信号を送信する．その際，最も出現数が多い宛先 IP アドレスも添える．エントロピーが低いほど，外部から届くパケットが特定の IP アドレス宛に集中していることが示唆される．

検知モジュールは機械学習モデルを用いて攻撃判定を行うものであり，Compute ノードにおいて動作する．エントロピーモジュールからトリガー信号を受信すれば，通知された IP アドレス宛のパケットを 3 秒間キャプチャし，特徴量の抽出，前処理を行った後，学習済みモデルで DDoS 攻撃か良性かを判定する．機械学習アルゴリズムとして，

ロジスティック回帰，ランダムフォレスト，ナイーブベイズの 3 つの手法が用いられた．

実験の結果，3 つの機械学習手法のうちロジスティック回帰が最も高い精度 99.97 %と，最も短い検出時間 0.45 ミリ秒を記録した．

この手法の長所は 2 点ある．1 つ目は，機械学習モデルによる判定を全ての通信に対して行う場合と比較し，リソース消費が抑えられる点である．エントロピー計算により外部から届くパケット宛先 IP アドレスに偏りを検知した場合に，最も出現数が多い宛先 IP アドレスに関する通信に絞って機械学習モデルによる判定を行うためである．2 つ目は，攻撃検知を単一ノード上に実装する場合と比較し，負荷分散が行えることである．単一のノードに攻撃検知手法の全てを実装するのではなく，複数のノードで分散して実装することで，単一ノードに攻撃検知のための負荷が集中しない．

一方で，パケットキャプチャ時間について，エントロピーモジュールと検知モジュールのそれぞれで 3 秒間行うことから，攻撃検知までに少なくとも 6 秒間かかるため，攻撃検知のリアルタイム性に課題がある．

4. 提案手法

本研究は，Patil らが提案したエントロピーと機械学習を用いた手法 [2] を土台とし，検知モジュールのパケットキャプチャ時間を性能を維持しながら最小化して検知時間を短縮することを目的とする．アーキテクチャ自体は踏襲し，キャプチャ時間を段階的に短縮して攻撃検知性能を維持できる最小のキャプチャ時間を調査する．

図 1 にエントロピーモジュールと検知モジュールの連携フローを示す．

検知性能を保ったままキャプチャ時間 T_{cap} を最小化し，検知モジュールの合計処理時間 T_{total} を短縮する．

$$T_{\mathrm{total}} = T_{\mathrm{cap}} + T_{\mathrm{flow}} + T_{\mathrm{other}}$$

ここで T_{flow} はフロー抽出にかかる時間，T_{other} はその他の推論などにかかる時間である．

再現性確保のため，本研究の実装では代表的なフロー抽出ツールである CICFlowMeter[*1] を採用した．ただし，本研究の主張はキャプチャ時間の最小化にあり，抽出ツールに依存しない．

原著アーキテクチャを保ちつつ，検知モジュール側のキャプチャ時間のみを変更した．キャプチャ時間を段階的に設定し，各時間につき 10 回反復して以下の 3 点を記録した．

- 検知成功回数
- 合計処理時間
- 抽出フロー数

[*1] 入手先 ⟨https://github.com/UNBCIC/CICFlowMeter⟩

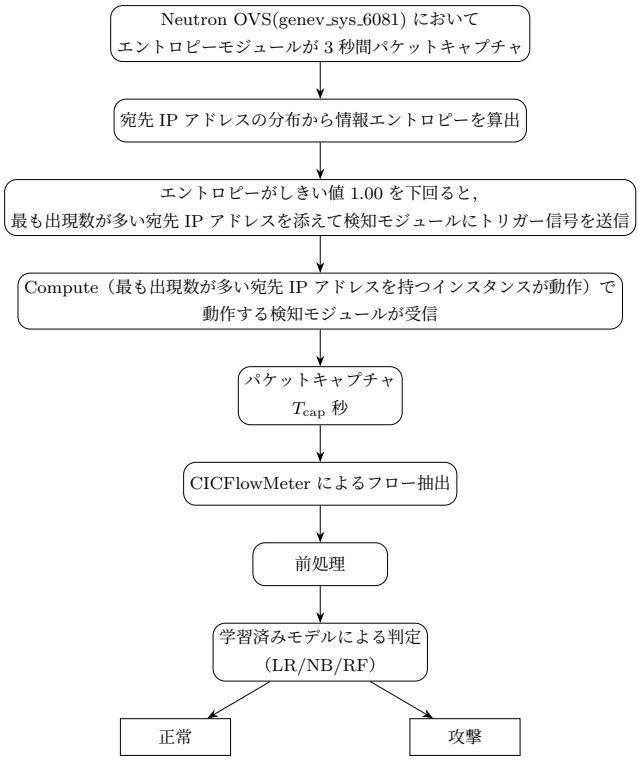

図 1 検知アーキテクチャの流れ（T_{cap} を短縮）

あるキャプチャ時間で 10 回試行し，その全てで攻撃が検知できることを「成功」と定義する．モデル別に攻撃検知性能を維持できる最短キャプチャ時間を調査する．

5. 実験設定

5.1 実験環境

表 2 実験環境の仕様

項目	構成
Controller	コントロールサービス（Keystone, Nova-API など）
Network(Neutron)	ネットワークサービス（Neutron）OVS: `br-int`，`br-ex`，トンネル用 `genev_sys_6081`
Compute1，2	コンピュートサービス（Nova）
仮想スイッチ	Open vSwitch
抽出ツール	CICFlowMeter（フロー統計量抽出）
機械学習	LR / NB / RF（scikit-learn），乱数シードは固定

本研究は表 2 のように，OpenStack 4 ノード構成（Controller，Network(Neutron)，Compute1，Compute2）の環境を使用した．各ノードの OS は CentOS Stream 9, OpenStack バージョンは Caracal，仮想ネットワークは Open vSwitch (OVS) により構成し，トンネル方式は Geneve（ポート 6081）を用いた．外部接続は Network ノードの `br-ex` 経由，テナント内は `br-int`–トンネル用ブリッジ間でカプセル化転送される．

図 2 に実験環境の論理トポロジを示す．Controller ノー

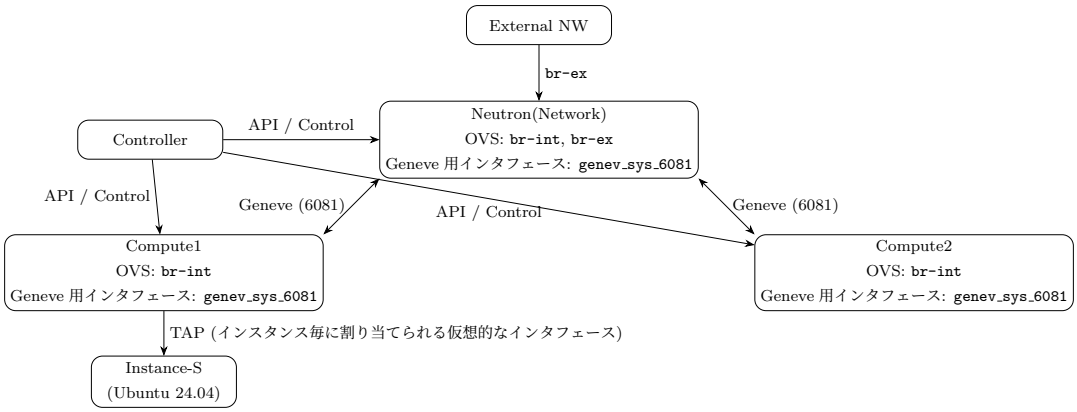

図 2 実験環境の論理トポロジ（Neutron でエントロピーモジュール，Compute1 で検知モジュールが動作する）

ドは各ノードに対し，OpenStack の API を用いた制御を行うものである．Compute1 および 2 ノードで動作するインスタンスと External NW との通信は Neutron ノードを経由して行われる．攻撃および良性通信のシミュレーションは External NW にある攻撃送信用の物理マシンから Compute1 上の Instance-S に向けて行う．

Instance-S の OS は Ubuntu 24.04 Server クラウドイメージであり，Apache による Web サーバも実行する．

Neutron ノードではエントロピーモジュールが動作する．Geneve インタフェースでパケットをキャプチャし，宛先 IP アドレスの出現数についてエントロピー計算を行う．エントロピーがしきい値を下回る場合に，最も出現数が多い宛先 IP アドレスを添えて Compute1 ノード上の検知モジュールに対しトリガー信号を送る．

Compute1 ノードでは検知モジュールが動作する．Geneve インタフェースで，エントロピーモジュールから通知された宛先 IP アドレスでフィルターを設定し，パケットキャプチャする．キャプチャデータを CICFlowMeter に入力してフロー統計量を生成し，前処理を行い，学習済み機械学習モデルで攻撃か否かを判定する．

5.2 評価指標と成功条件の定義

本研究では，検知モジュールがエントロピーモジュールからのトリガー受信後に攻撃／良性を最終判定するまでの合計処理時間 T_{total} を主指標とし，分解指標としてパケットキャプチャ時間 T_{cap}，フロー抽出時間 T_{flow}（特徴量抽出・前処理を含む）を計測する．あわせて抽出フロー数（CICFlowMeter が生成したフローを前処理した後に残ったフローの数）を記録する．成功条件は，各キャプチャ時間設定に対して 10 回試行の全てで攻撃検知が成立した場合を「攻撃検知成功」と定義する．

5.3 データセットの作成

良性データは HTTP GET リクエストとレスポンスのトラフィックから作成した．具体的には，Compute1 上にある Instance-S で動作する Web サーバに対し，External NW にある物理マシンから 30 分間，0.5 秒間隔で HTTP GET リクエストを送信することでトラフィックを発生させた．同時に，Compute1 上で tcpdump を用いてキャプチャし，CICFlowMeter によりフロー抽出を行った．

攻撃データは SYN-Flood 攻撃のトラフィックから作成した．具体的には，External NW にある物理マシンから Instance-S の TCP ポート 80 番に対し，SYN Flood 攻撃を行った．良性データと同様にキャプチャし，フロー抽出を行った．

5.4 前処理

生成した良性・攻撃データセットを 1:1 で結合してシャッフルした後，環境依存の列（Flow ID, Src/Dst IP, Src/Dst Port, Timestamp）および分散が 0.0 の 22 項目（Protocol, Fwd Packet Length Min, Bwd Packet Length Min, Fwd PSH Flags, Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags, Packet Length Min, URG Flag Count, CWR Flag Count, ECE Flag Count, Fwd Bytes/Bulk Avg, Fwd Packet/Bulk Avg, Fwd Bulk Rate Avg, Bwd Bytes/Bulk Avg, Bwd Packet/Bulk Avg, Bwd Bulk Rate Avg, Active Mean, Active Std, Active Max, Active Min, Idle Std）を除外した．その後，学習データとテストデータに 7:3 の割合で分割した．

5.5 モデルの学習

学習済みモデル（ロジスティック回帰，ナイーブベイズ，ランダムフォレスト）を作成した．テストデータを用いて評価を行った結果，精度はいずれも 100 %になった．

5.6 キャプチャ時間の設定と比較設定

検知モジュールのキャプチャ時間 T_{cap} を 3 秒から段階的に短縮し，各設定につき 10 回試行する．モデル毎に成

功可否（10/10 成立）・T_{total}・T_{flow}・抽出フロー数を記録する．

5.7 検知モジュールにおける前処理

検知モジュールでは，データセット作成時に行った前処理に加え，以下の処理を行った．

(1) **SYN Flag Count が 0 のフローを削除:** キャプチャの開始前からすでに開始されている TCP コネクションのフローを除外するため．

(2) **FIN Flag Count が 2 未満かつ RST Flag Count が 0 のフローを削除:** キャプチャ終了時にまだ終了していない TCP コネクションのフローを除外するため．

5.8 時間分解計測の方法

Python の time.perf_counter()（ナノ秒精度）で基準時刻を取得し，合計処理時間 T_{total}，パケットキャプチャ時間 T_{cap}，フロー抽出時間 T_{flow}（特徴量抽出・前処理を含む）を計測する．

5.9 各キャプチャ時間について攻撃検知可否を調査

実験の条件は以下のとおりである．

- ロジスティック回帰，ランダムフォレスト，ナイーブベイズの機械学習モデルについてそれぞれ検証を行った．
- 検知モジュールのキャプチャ時間を 3 秒から段階的に減少させ，各キャプチャ時間について攻撃検知を試行した．
- 攻撃トラフィックとして，DDoS 攻撃データセット作成時と同様のものを発生させた．
- 各キャプチャ時間について 10 回の試行を行い，その全てで攻撃検知が成功する場合を「攻撃検知成功」と定義した．
- 試行ごとに，抽出フロー数，フロー抽出に要した時間，トリガー信号受信から攻撃検知までの合計処理時間を記録した．

6. 実験結果

結論から述べると，ランダムフォレスト（RF）では $T_{\text{cap}} = 0.237$ 秒 まで短縮しても検知成功（10/10）を維持し，そのときの合計処理時間 T_{total} は 1.83 秒であった．これは従来の 3.000 秒キャプチャ時の 6.999 秒に対し 5.169 秒の短縮である．

6.1 キャプチャ時間別の処理時間および検知回数

各モデル（LR/NB/RF）に対するキャプチャ時間 T_{cap} を段階的に短縮させて試行した結果を表 3，4，5 に示す．なお，抽出フロー数，T_{flow} および T_{total} は，推論に至った場合における平均値である．つまり，抽出フロー数が 0 になり，推論を行わなかった場合を含まない．また，抽出フロー数とは，CICFlowMeter が生成したフローから先述の前処理を行い，最終的に残ったフローの数のことである．

表 3 LR に対するキャプチャ時間別の結果（各時間 10 回試行）

T_{cap} [s]	抽出フロー数	T_{flow} [s]	T_{total} [s]	検知回数
0.300	4.33	1.410	1.948	9
0.305	4.56	1.390	1.928	9
0.306	6.00	1.388	1.934	9
0.307	5.20	1.434	1.987	10
0.310	6.80	1.384	1.946	10
0.320	4.80	1.395	1.963	10
0.330	4.80	1.367	1.959	10
0.340	4.10	1.412	2.021	10
0.350	9.60	1.400	2.040	10
0.400	9.20	1.402	2.135	10
0.500	9.80	1.444	2.320	10
1.000	25.60	1.487	3.312	10
2.000	89.20	1.742	5.139	10
3.000	198.80	1.975	7.036	10

表 4 NB に対するキャプチャ時間別の結果（各時間 10 回試行）

T_{cap} [s]	抽出フロー数	T_{flow} [s]	T_{total} [s]	検知回数
0.500	11.00	1.422	2.297	4
0.700	18.60	1.462	2.655	9
0.705	14.00	1.455	2.670	10
0.706	18.10	1.446	2.683	10
0.710	15.50	1.404	2.647	10
0.720	18.90	1.461	2.694	10
0.730	17.80	1.451	2.722	10
0.740	17.00	1.460	2.719	10
0.750	16.30	1.427	2.719	10
0.800	19.90	1.474	2.850	10
0.900	29.90	1.547	3.176	10
1.000	30.80	1.529	3.324	10
2.000	100.80	1.781	5.209	10
3.000	192.10	1.988	7.041	10

表 5 RF に対するキャプチャ時間別の結果（各時間 10 回試行）

T_{cap} [s]	抽出フロー数	T_{flow} [s]	T_{total} [s]	検知回数
0.200	2.25	1.393	1.780	8
0.230	2.89	1.375	1.832	9
0.235	4.33	1.343	1.787	9
0.236	4.67	1.372	1.830	9
0.237	2.90	1.391	1.830	10
0.240	3.80	1.376	1.829	10
0.250	4.30	1.360	1.832	10
0.300	4.50	1.451	2.011	10
0.400	5.30	1.392	2.115	10
0.500	8.60	1.409	2.291	10
1.000	33.70	1.546	3.396	10
2.000	93.90	1.724	5.113	10
3.000	193.00	1.951	6.999	10

6.2 処理時間の分解比較

3 つのモデルの中で最も短い T_{cap} で検知成功した RF に対して, T_{cap} を段階的に短縮したときの T_{flow}, T_{other}, T_{total} の平均（10 回）を表 6 に示す．キャプチャ時間の短縮により，フロー抽出時間や推論を含むその他の処理時間も減少した．最短キャプチャ時間の 0.237 秒 では $T_{\mathrm{total}} = 1.83$ 秒 となり，合計処理時間は基準（3.000 秒）の 6.999 秒から 5.169 秒短縮できた．

表 6 RF における処理時間の内訳（平均，単位：s）と短縮量

T_{cap}	T_{flow}	T_{other}	T_{total}	T_{total} の短縮量
0.237	1.391	0.202	1.830	5.169
0.240	1.376	0.213	1.829	5.170
0.250	1.360	0.222	1.832	5.167
0.300	1.451	0.260	2.011	4.988
0.400	1.392	0.323	2.115	4.884
0.500	1.409	0.382	2.291	4.708
1.000	1.546	0.850	3.396	3.603
2.000	1.724	1.389	5.113	1.886
3.000	1.951	2.048	6.999	–

6.3 結果の要約

本実験結果の要点を表 7 に整理する．RF は 0.237 秒で 10/10 成立かつ $T_{\mathrm{total}} = 1.83$ 秒を達成し，3.000 秒基準に対して 5.169 秒短縮した．この結果は，エントロピー側の 3 秒周期トリガー内に検知モジュールの処理を収められることを示し，モジュール同士がより協調的な動作すると考えられる．

表 7 最短キャプチャ時間の比較と主要指標

モデル	最短 T_{cap} [s]	T_{total} [s]（そのとき）	T_{total} 短縮量
LR	0.307	1.987	5.049
NB	0.705	2.67	4.371
RF	0.237	1.83	5.169

検知性能を保った最短キャプチャ時間はモデルに依存し，RF が最短のパケットキャプチャ時間と最も大きな遅延短縮を実現した．推論計算は全体遅延に与える影響が比較的に小さいと考えられ，キャプチャ時間とフロー抽出時間の最小化が鍵である．

7. 考察

7.1 結果の解釈

本研究では，Patil らのエントロピーと機械学習を用いた手法 [2] を踏襲しつつ，検知モジュールのパケットキャプチャ時間 T_{cap} を短縮しても検知性能（10/10 成立）を維持できることを示した．特にランダムフォレスト（RF）において $T_{\mathrm{cap}} = 0.237$ 秒 まで短縮しても検知性能は低下せず，合計処理時間 $T_{\mathrm{total}} = 1.83$ 秒 に短縮することができている．これは $T_{\mathrm{cap}} = 3.000$ 秒 時の $T_{\mathrm{total}} = 6.999$ 秒 と比べて 5.169 秒 の短縮であり，検知モジュールの処理時間が，エントロピーモジュールのトリガーの最短周期である 3 秒に収まることで両モジュールがより協調的に動作することを示唆する．

処理内訳（表 6）からは，キャプチャ時間の短縮によって T_{cap} のみならずフロー抽出時間 T_{flow} や推論を含むその他の処理時間 T_{other} が減少したことが確認できる．

7.2 10 回のうち数回だけ攻撃を検知できたケース

攻撃検知が複数回行われたものの，10 回に至らなかったケースの原因として 2 つが考えられる．

1 つ目は，攻撃判定を行うために必要なデータ量が不足していることである．短時間にするほどキャプチャできるパケットが減少し，生成され，前処理を行い最終的に残るフロー数も少なくなる．実際に，フローが全く得られない場合もあった．

2 つ目は，機械学習モデル間の性能差である．ランダムフォレストがキャプチャ時間 0.237 秒，平均抽出フロー数が 2.9 個でも攻撃を 10 回中の全てで検知可能であるのに対し，ナイーブベイズではキャプチャ時間 0.7 秒，平均抽出フロー数が 18.6 個で 10 回中 1 回攻撃検知に失敗している．アルゴリズムの特性の違いや，学習方法により，モデルの間に性能差が生じたと考えられる．

7.3 エントロピーモジュールのトリガー周期との関係

エントロピーモジュールは最短 3 秒周期でトリガーを発し，原著では検知モジュールでも 3 秒間のキャプチャを想定していたため，エントロピーモジュールの動作も含めた検知までにかかる処理時間は，少なくとも 6 秒 を要するという課題があった．本研究は検知モジュールにおける合計処理時間を短縮し，$T_{\mathrm{total}} < 3$ 秒 を満たした点に実装上の意義がある．エントロピーモジュールも含めた合計処理時間をさらに短縮するためには，キャプチャやエントロピー計算方法について改良が求められる．

7.4 課題

本研究の課題を以下に整理する．

- **攻撃の種類が限定されている**：評価は SYN Flood を対象とし，攻撃発生源は単一マシンに限定した．より大規模なボットネット型やアプリ層（HTTP）フラッド等への一般化は追加の検証が必要である．
- **短期的な特徴量**：通信を瞬間的に切り取って特徴量抽出を行っているため，Slowloris のような長時間の通信を伴う攻撃の検知への応用が難しいと考えられる．
- **直列処理の限界**：パケットキャプチャからモデルによる判定までの処理をほとんど直列で行っているため，それぞれの処理が完全に終了するまで次の処理に移る

ことができない課題がある．高速化には，並列的な処理を行うことが求められる．

8. おわりに

本研究は，OpenStack ベースのクラウド環境における DDoS 攻撃（SYN Flood）検知に対し，エントロピーと機械学習を組み合わせた既存のアーキテクチャを踏襲しつつ，検知モジュールのパケットキャプチャ時間の最小化に焦点を当てて検知までの合計処理時間を短縮する手法を提案・検証した．キャプチャ時間を段階的に短縮し，ロジスティック回帰（LR），ナイーブベイズ（NB），ランダムフォレスト（RF）の 3 モデルで評価した結果，検知成立（各条件 10/10 回）を維持したままの最短キャプチャ時間を縮めることができた．とくに RF では $T_{\mathrm{cap}} = 0.237$ 秒時に合計処理時間 $T_{\mathrm{total}} = 1.83$ 秒を達成し，従来の $T_{\mathrm{cap}} = 3.000$ 秒時の $T_{\mathrm{total}} = 6.999$ 秒と比べて 5.169 秒の短縮を実現した．この短縮により，検知モジュールの処理時間がエントロピーモジュールの 3 秒周期のトリガー送信に収まることが示され，両モジュールの連携動作向上と迅速な攻撃検知に寄与する．

処理時間の分解から，キャプチャ時間とフロー抽出時間が遅延の支配要因であることを確認した．したがって，検知の高速化にはキャプチャ制御と特徴量抽出の軽量化／並列化が最も効果的であると考える．また，本研究の主張はキャプチャ時間に依存しており，フロー抽出ツールに対しては本質的に非依存である（実装では再現性のため CICFlowMeter を使用）．推論を含むその他の処理時間について，具体的にどのような処理に時間がかかるのかについてより詳細な分析を行う必要がある．

評価は SYN Flood・単一サイト・特定の仮想ネットワーク構成に限定しており，クラス比 1:1 の前提も実運用とは異なる可能性がある．誤検知率を抑えた実運用適用には，多様なトラフィックが発生する環境下での再評価，運用閾値の最適化，他の特徴量抽出方法での検証が必要である．

総括すると，本研究は検知性能を維持した最短キャプチャ時間の同定と，それに基づく検知遅延の顕著な短縮を実環境（OpenStack）で示した．提案は既存アーキテクチャに対し侵襲性が低く，交換可能な構成要素（学習器・抽出器・仮想化方式）と組み合わせやすい．今後は，UDP ／ ICMP ／ HTTP 等の多様な DDoS シナリオや高負荷下での再評価を進める．あわせて，エントロピーモジュール側の改良と組み合わせた検知遅延のさらなる短縮を目指すことで，クラウド基盤の実運用に耐える迅速な DDoS 検知の実現に貢献したい．

参考文献

[1] 株式会社インターネットイニシアティブ：Internet Infrastructure Review（IIR）Vol.66, 2025 年 3 月, 入手先 ⟨https://www.iij.ad.jp/dev/report/iir/066/01.html⟩（参照日：2025.08.16).

[2] R. Patil et al.：“A Collaborative Approach to Detect DDoS Attacks in OpenStack-based Cloud using Entropy and Machine Learning” 2023 14th Intl. Conf. on Computing Communication and Networking Technologies (ICCCNT), 2023, pp. 1-5.