Vol.2019-DPS-178 No.11
Vol.2019-CSEC-84 No.11
2019/3/4

IoTネットワークのためのイベント駆動による ネットワーク異常検出・対処システムの提案と評価

木全　崇[1,a)]　寺西　裕一[1,b)]　細川　貴史[1,c)]　原井　洋明[1,d)]

概要：将来のIoT(Internet of Things)においては，デバイス(カメラ，センサ，ロボット，車など)が生成するセンサーデータ等がネットワークを通じて送受信される．こうしたデータがプライバシ情報等を含む場合，通常は専用のネットワークが用いられる．しかし，IoTでは，システムの物理的な隔離が困難であるため，盗聴機器を仕掛ける攻撃や，マルウェア感染した計算機の接続による攻撃のリスクが高まる．また，単機能のデバイス上での異常検出や対処は，性能上の制約により困難な場合が多い．こうした課題に対し，ネットワーク側で監視・対処する方法もあるが，監視データの収集や処理の負荷が大きく，また，対処に時間がかかってしまう課題がある．本稿では，トラヒック変動等のイベントを分析する機能をネットワーク機器上に分散配置することで異常を素早く検出し，当該箇所を仮想的に切り離し可能とするシステムを提案する．シミュレーションおよび実機を用いた評価により，提案システムが再現率100%を維持しつつ，適合率を90%以上で想定異常イベントを検出でき，かつ，概ね3秒以内で異常検知・対処が可能となることを確認した．

キーワード：IoT，ネットワーク異常検知，イベント駆動

Proposal and Evaluation of An Event-Driven Anomaly Detection and Recovery System for IoT Networks

Takashi Kimata[1,a)]　Yuuichi Teranishi[1,b)]　Takafumi Hosokawa[1,c)]　Hiroaki Harai[1,d)]

Keywords: IoT, Network Anomary Inspection, Event-Driven

1. はじめに

IoT(Internet of Things)においては，デバイス(カメラ，センサ，ロボット，車など)がネットワーク接続し，センサーデータ等を常時送受信する．センサーデータはプライバシ情報等を含むことが多く，機密維持のため一定のセキュリティレベルがネットワークに求められる．

一般に，ネットワークを通じた遠隔からのシステムへの侵入等の攻撃は，システムを論理的に分離された仮想ネットワーク上に隔離して構成し，その入出力を担うゲートウェイ部分での入出力制限や監視によって回避できる．

一方，物理的な攻撃，すなわち，ネットワーク機器やケーブル等に対して攻撃者が直接的・物理的に改変を行い，盗聴機器を仕掛ける攻撃や，マルウェア感染した計算機を物理的に接続することによる攻撃に対する対策も重要性が高い．一般に，ICTサービスを構成するシステムは，入退室が制限されたデータセンター等の中に構築され，物理的なセキュリティが確保される．しかし，IoTシステムを構成するIoTネットワークは，センサー等が建物内などに設置され，物理的に一般ユーザが近づくことができる位置に構成されるため，物理的な隔離が難しい．

[1] 情報通信研究機構
4-2-1, Nukui-Kitamachi, Koganei, Tokyo 184–8795, Japan
[a)] kimata@nict.go.jp
[b)] teranisi@nict.go.jp
[c)] takafumihosokawa@nict.go.jp
[d)] harai@nict.go.jp

こうした物理的な攻撃の検出や，攻撃に対する対処を行う方法として，末端機器等が対策機能を持つソフトウェア機能を具備することが考えられる．人が利用するパソコン等の機器にマルウェア対策機能などを具備させる対策は一般的である．しかし，IoT ネットワークを構成するセンサー等のデバイスは，安価な機器が用いられることが多く，そのようなソフトウェア機能を実行することが性能上の制約で困難な場合が多い．このため，ネットワーク上，あるいは，クラウド上で攻撃を検知・対処する方法が数多く検討されてきた．

ネットワーク上で，物理的な構成変更を異常として検出するには，攻撃にともなう通信状況の変動を捉えなければならない．すなわち，機器からどの機器へ，どのような頻度でデータが送信されているか，といったネットワークレイヤのトラヒック (以下，ネットワークトラヒック) の変動を捉える必要がある．ネットワーク機器上に従来から実装されている Simple Network Management Protocol (SNMP) [1] は，物理的なポートやインタフェースの状態（データリンクレイヤ）の監視や，累積パケット数等の集計値は取得できるが，上記ネットワークトラヒックは把握できない．従来，ネットワークトラヒックの分析を行う手段としては，IDS(Intrusion Detection System) 等を用いて，パケットのペイロードを含めて分析する DPI (Deep Packet Inspection) が広く用いられてきた．しかし，IDS はネットワークの入出力部分に個別に設置する必要があり，複数のセンサーネットワークを含む IoT ネットワークでは取得できる情報に限りがある [2]．また，個別データパケットのペイロードの再構成を行なうため，通信性能への影響は避けられない．

こうした課題に対処するため，データパケットのペイロードは分析せず，ネットワークトラヒックを小さい処理負荷で把握可能とするプロトコルとして，Internet Protocol Flow Information eXport（IPFIX）[3]（あるいは NetFlow [4]．以下，NetFlow と表記）が標準化され，多くのネットワーク機器で利用可能となりつつある．NetFlow に基づく異常検出手法も数多く提案されている [5] [6]．しかし，NetFlow を用いたネットワークトラフィックの分析処理は，ネットワークトラヒック (Flow) 情報を一旦集約し，集約先で分析処理を実行する形態が基本である．すなわち，集中管理型のアーキテクチャが想定されており，Flow 数が多いと集約にともなうネットワーク負荷・分析処理負荷が高くなり，頻度の高い異常分析処理は実行できず，対処に時間がかかってしまう課題がある．

上記を鑑み，本研究では，大量のネットワークトラヒック情報の集約を行なわず，物理的な攻撃によるネットワークトラヒックの異常をネットワーク機器上でイベントとして検出し，素早い対処を可能とするシステムを提案する．本研究の貢献は次の通りである．

- IoT ネットワークにおいて，物理的な攻撃に関連するネットワークトラヒックの異常状態をイベントとして定義した．また，当該イベントの発生を NetFlow 情報に基づいて検出・通知する機能をネットワーク機器上に分散配備し，自動的に隔離対応するシステムを提案した．
- 提案システムを，NetFlow 情報に基づく on-box programming が可能なスイッチを用いて実装した．また提案システムの性能を，IoT ネットワークを想定したシミュレーション及び実機により評価した．その結果，性能を保ちつつ，十分な応答速度・精度で想定される異常検出・対処が可能であることを示した．

2. IoT ネットワークにおけるトラヒック異常イベント

IoT ネットワークにおいて想定される物理的な攻撃のうち，特に検出が重要となるものとして以下が挙げられる．

（1） 物理リンクの切断（異常停止）・再接続
（2） 意図しない動作をする端末の物理的な接続

本節では，これらに対応するネットワークトラヒックの変動をイベントとして定義し，それらの検出方法について議論する．

2.1 Removal Anomaly

（1）は，敷設されている LAN ケーブルをネットワーク機器から抜線，あるいは切断し，盗聴機器等を追加して再接続することを指す．このようにして生じる切断状態を Removal Anomaly (RA) イベントと定義する．

一般的な TCP/IP を用いるアプリケーションでは，LAN ケーブルが抜線された物理的な位置によっては，コネクションがリセットされる等の現象が起き，アプリケーションレベルでの障害として検出できる．しかし，物理的な位置によっては数秒から数分のタイムアウトが生じるまで検出できず，また，タイムアウト前に再接続されると，異常として検出できない．

RA は，ケーブルが物理的に一定時間，ネットワークから分断された状態となるため，IoT ネットワークを構成するすべてのネットワーク機器が物理リンクの状態変化を通知できれば，検出できる．しかし，物理リンクの状態に基づく RA の検知・通知には，物理インタフェース状態監視機能，管理ネットワークを通じた SNMP 機能等の実装が必要である．したがって，安価な末端のネットワーク機器を含む IoT ネットワークの構成機器すべてが具備することは期待できない．

一方，RA ではネットワークトラヒックが流れない時間が生じるため，IoT ネットワークにおけるセンサーのよう

表 1　トラヒック異常イベント種別

名称	定義	原因
Removal Anomaly (RA)	想定トラヒックの停止	LAN ケーブルの抜線・切断
Addition Anomaly (AA)	想定外トラヒックの発生	新規端末の追加 (DoS)
Scan Anomaly (SA)	接続先数の異常増加	ポートスキャン (DDoS)

に常時データが生成・送信され，クラウドに送信されている前提の元ではネットワークの上流部分に単位時間あたりに想定されるネットワークトラヒック量が生成されない，あるいは，全くトラヒックがない状態となった時点で，検出可能である．センサーネットワークでは，一定間隔で周期的にデータ送信が行われる想定が可能な場合が多く，その場合，生成・送信される頻度が少ない場合でも観測を行う時間がデータ生成周期より長ければ検出できる．

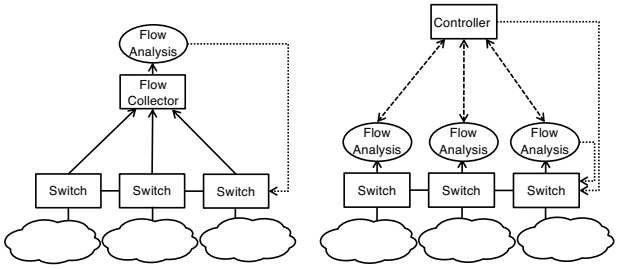
2.2 Addition Anomaly

（2）としては，まず，攻撃用の端末を新たに IoT ネットワークに接続し，攻撃を行う，あるいは，新規接続される端末を乗っ取って攻撃を行うことが該当する．後者は，例えば，新規接続センサーが接続前からマルウェア感染していた（あるいは，接続後に感染した）状態などで生じる．

通常，使用しないネットワーク機器のインタフェース（物理ポート）を無効としておくことで，前者の新たな端末の接続は防ぐことができる．また，RA と同様，すべてのネットワーク機器が物理リンクの状態変化をイベントとして検出できれば，想定外の端末がネットワーク接続された時点で検知できる．しかし，IoT ネットワークの構成機器すべてにこれらの対策が施されることは，期待できない．また，後者の，感染端末の接続の場合は，物理リンク状態が変化すること自体が異常状態ではないため検出できない．

一方，攻撃が DoS(Denial of Service) であった場合，新たなネットワークトラヒックが生成される．すなわち攻撃端末によって，一定数・容量のネットワークトラヒックが新たに生じる．このように生じる異常状態を Addition Anomaly(AA) イベントと定義する．

センサーネットワークでは，デバイス上でのデータ生成頻度等があらかじめ定められるため，生成されるトラヒックを予見できる．したがって，センサーネットワークにおける AA の多くは検出可能である．すなわち，設定した送信元から送信先への単位時間あたりのネットワークトラヒック量が，設定以上に生じた，あるいは，設定外の送信元から，あるいは設定外の送信先へ，新たにネットワークトラヒックが生じた時点で検出可能である．しかし，人が端末を用いる場合（ウェブブラウジング等），ネットワークトラヒック量のみでの判別は困難となる．



(a) NetFlow ベースシステム　(b) 提案システム

図 1　ネットワークトラヒック分析・処理のアーキテクチャ

2.3 Scan Anomaly

（2）としては，DDoS(Distributed Denial of Service) を目的として，他のネットワーク内の端末を乗っ取る動作を行う端末の接続も該当する．この場合，乗っ取り可能な端末を調査するため，ポートスキャンあるいはそれに類する動作が行われる．すなわち，トラヒック量としては小さいが，通常の活動では見られない数の送信先との間で通信を試みる．このように，ネットワーク内に存在する端末の調査を行うネットワークトラヒックが生成される状態を Scan Anomaly(SA) イベントと定義する．

SA は，意図しない新たな送信元から，あるいは，新たな送信先へのネットワークトラヒックが生じた時点で，AA と同様の方法で検出が可能である．しかし，想定される範囲内での通信の場合，すなわち，IoT ネットワークを構成する機器間での通信が常時行われる状況では，送信元や送信先の限定が困難となる．一方，ポートスキャンでは，通常よりも多い数の相手との間での通信を試みることになると考えられる．したがって，そのような状態となったことが観測された時点でイベントとして検出することは可能である．

3. 提案システム

我々は，前節で示した IoT ネットワークにおけるネットワークトラヒックの異常イベントを，ネットワーク機器上で検出・対処するシステムを提案する．本節では，提案システムのアーキテクチャおよび実装について述べる．

3.1 アーキテクチャ

図 1 は，提案システムのアーキテクチャ (図 1(b)) を既存の NetFlow ベースのシステム (図 1(a)) との対比により示したものである．

既存システムでは，Flow Collector と呼ばれる部分でネットワークトラヒック情報 (Flow) を集約し，分析 (Flow Analysis) を行ったのち，必要に応じてネットワーク機器 (Switch など) を制御する形態が基本となる．Flow データは常時トラヒックが流れているとき，頻繁に生成されるため，ネットワーク負荷，および分析処理の負荷を軽減する方策が必要となる．通常はサンプリングされた Flow デー

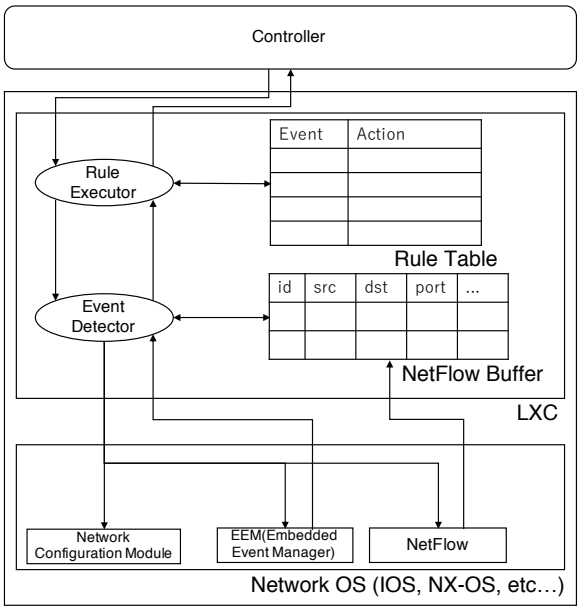


図 2　システム構成

タを定期的に収集する．

提案システムは，ネットワークトラヒックの分析を，従来のようにネットワーク機器外で分析するのではなく，ネットワーク機器上 (on-box) で実行するアーキテクチャをとる．上記アーキテクチャにおいては，システム全体を制御する Controller からの指示に従って，Switch 等のネットワーク機器上で Flow の分析を行なう．分析の結果，イベントとして検知された場合，当該イベントが検知されたことを Controller に通知する．異常イベントが検知され，ネットワーク機器上で対処が必要と判断できる場合は，Controller にイベントとして通知するほか，直接 Switch 等を制御し隔離・停止等の対処をすることも可能とする．これによって，異常イベントに対する迅速な対処ができる．

近年のネットワーク機器の進化にともない，提案システムのアーキテクチャは現実的なものとなってきており，普及価格帯のネットワーク機器であっても，Linux コンテナ等を用いた on-box programming や NetFlow 情報の参照が可能となりつつある．本研究では，そのような on-box programming 可能なネットワーク機器を用いて，提案アーキテクチャに基づくシステムを実装し，実現性を確認した．

3.2　実装

本節では，我々が実施した提案システムの実装について述べる．図 2 は，提案システムのネットワーク機器上の機能構成を示している．提案システムは，on-box programming 環境を提供するネットワーク機器上で動作するプログラムモジュールとして動作する．実装には，Cisco Systems 社製 L3 スイッチ Catalyst 9300 を用いた．本スイッチは，NetFlow 情報の利用や イベント処理を行なうための API を提供しており，同 API を用いたプログラムを Linux Containers(LXC) [7] 上で動作させることができる．以下で各機能要素について説明する．

図中の Rule Executor は，ルールの管理・実行を行なう機能要素である．ルールは，イベントと，イベントに対応して実行する実行命令（Action: プロセス起動・停止など）の組の集合であり，Controller から与えられる．Action としては，当該ネットワーク機器でイベントに対する対応処理を実行せず Controller へ通知 (pass-through) する動作もある．この場合，イベントに対する処理は Controller にて実行する．

Rule Executor に含まれるイベントは，Event Detector において，次の通り検知する．まず，Network OS として IOS [8], NX-OS [9] 等が有する Embedded Event Manager(EEM) [10] と呼ばれる機能を用いることで，例えば，SNMP MIB [11] のある値が閾値を越える場合に発生するイベント，ルーティングテーブルの変化を表すイベントなどを検知させる．これらイベントは，Rule Executor に渡される．RA イベント，AA イベントのうち，当該スイッチがもつ物理ポートへの端末の接続・切断が原因となるものは，この EEM によって検出可能である．

また，SA イベント，あるいや RA, AA イベントのうちネットワークトラヒックの分析が必要となるものについては，NetFlow 情報 (フロー ID, 送信元アドレス, 送信先アドレス，ポート番号, 入力インターフェース などから構成される) を取得し，一定の観測期間バッファ（Flow Buffer）に蓄積したうえで分析処理を行なうことで検知する．EEM では，スイッチの各ポートのトラフィックの流入量の変動が閾値を超えたことを検出できる．本実装では，AA や SA のイベント検出において，このトラヒック検出がなされた場合のみ NetFlow 情報のバッファへの蓄積を開始することで，無駄な NetFlow 情報取得のオーバヘッドを削減している．

Action に相当する動作は，on-box programming によって実行できる Python プログラミング言語を用いて実装した．IoT ネットワークにおいて AA, RA などが起きた場合の対処に対応する Action として，Network OS の Network Configuration API を用いて，Virtual Routing and Forwarding (VRF) [12] による仮想ネットワークの分離を指定可能とした．

3.3　イベント分析処理

本実装においては，Event Detector にて，NetFlow 情報を用いて，RA, AA, SA の各イベント検出処理を行う．RA は，次が成立するかどうかの判定により検出する．

$$\frac{\Delta D}{W} \leq T_{min}$$

ただし，W は，観測期間 (ウィンドウサイズ)，ΔD は，先の観測時刻以後に観測されたトラヒック (観測期間におけ

る転送データ量 (バイト数))，T_{min} は，下限値となる利用帯域 (単位時間あたり転送データ量（バイト数)) のしきい値である．上記が成立する場合，想定されるトラヒックが生じていないため，異常と判断する．

AA は，次を満たすかどうかの判定により検出する．

$$T_{max} \leq \frac{\Delta D}{W}$$

T_{max} は，上限値となる利用帯域のしきい値である．上記が成立する場合，想定される以上のトラヒックが生じているため，異常と判断できる．IoT デバイスが停止しているとき，$T_{max} = 0$ である．

SA は，次が満たされるかどうかの判定により検出する．

$$C_{max} < \frac{\Delta P}{W}$$

C_{max} は，上限値となる単位時間あたりの通信相手数，ΔP は，先の観測時刻以後に観測された通信相手数である．上記が成立するならば，異常と判定する．単機能のセンサーであれば，データ送信先は基本的に 1 つであり，W がデータ送信周期と合致するとき，$C_{max} = 1$ となる．

$W, T_{min}, T_{max}, C_{max}$ は，それぞれ，どこから，どこへ，どのようなデータ転送が実行されるかによって適切な設定が変化する．したがって，本実装においては，IoT ネットワーク上でのセンサーの起動や停止にともなって，Controller を通じたルールの一部として通知する．

4. 評価

4.1 検出性能

提案システムの検出性能を，実際のネットワークトラヒックデータを用いたシミュレーションによって評価した．

4.1.1 評価尺度

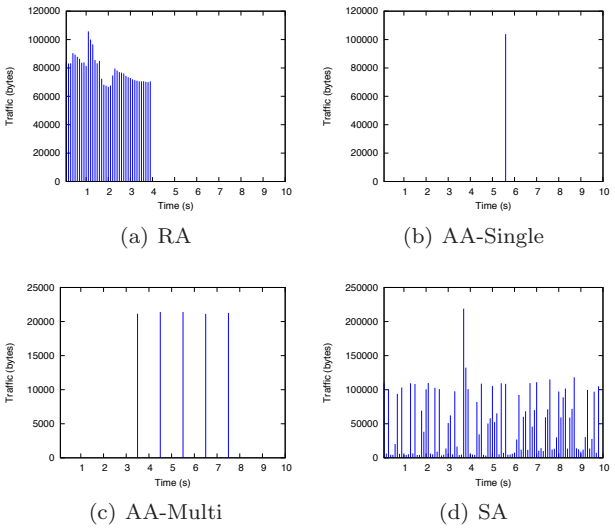
検出性能の評価尺度としては次を用いた．

- *precision*(適合率)
  検出されたイベントのうち，実際に発生した異常イベントの割合．
- *recall*(再現率)
  実際に発生した異常イベントのうち，検出された割合．
- *response*(応答時間)
  実際のイベント発生から，検出されるまでにかかる時間．

precision の定義は次の通りとした．

$$precision = \frac{TP_p}{TP_p + FP_p}$$

TP_p は，検出された異常パケットの数，FP_p は，検出されたが，異常パケットではなかった数である．一方，*recall* の定義は次の通りである．



(a) RA　(b) AA-Single　(c) AA-Multi　(d) SA

図 3 異常トラヒックパターンの例

$$recall = \frac{TP_e}{TP_e + FN_e}$$

TP_e は，検出された異常イベント（攻撃）の数，FN_e は，検出されなかった異常イベント（攻撃）の数である．

一般に *precision* と *recall* はトレードオフの関係にある．異常検知においては，*precision* は 1.0 に近い方が良いが，1.0 でないとしても，異常ではない状態が異常と捉えられたのみであり，それほど大きな問題ではない．一方，*recall* が 1.0 ではないとすると，異常が見逃されたことになり，問題としては大きい．このため，*recall* を 1.0 に保ちつつ *precision* を最大化できることが望ましい．

response は，実際に異常イベントが発生した時刻から，TP_e として観測された時刻までの差を表す．

4.1.2 データセット

提案システムの検出性能を調べるため，プライバシに関する情報が扱われる IoT ネットワークを想定した物理攻撃イベントを再現するパケットキャプチャデータを作成した．図 3 は，評価に用いたトラヒックパターン（抜粋）である．

図 3(a) は，カメラ映像を送受信するネットワークにおいて，データ送信中に断線する RA イベントの例である．この例は，フレームあたり約 80Kbyte の映像データを 10fps で送信中，図中の 4 秒目付近でネットワークの一部で断線状態となっている．

図 3(b),3(c) は，AA イベントの例である．図 3(b) は，ある送信元から，単独の攻撃トラヒックが生成される状況 (AA-Single と呼ぶ)，図 3(c) は，連続して攻撃トラヒックが生成される状況 (AA-Multi と呼ぶ) を再現している．

図 3(d) は，利用者によるデータ転送とポートスキャンが混在している SA イベントの例である．この例では，図中の，5 秒目付近から ポートスキャンが発生しており，小さいサイズのデータトラヒックが継続して生成されている．

表 2　シミュレーション設定

パラメータ	値
イベント検出周期	3（秒）
T_{min}	1000（バイト）
T_{max}	100（バイト）
C_{max}	50
試行回数	100

図は，ユーザ活動が混在する状況における侵入攻撃のデータセットである CICIDS2017 [13] からの抜粋である．評価では，CICIDS2017 のうち，侵入を伴う DDoS 攻撃が行われた「Thursday, July 6, 2017」のデータ（8 時間にわたる約 5GB のデータトラヒックを含む）を用いている．このデータセットでは，NAT ゲートウェイ配下に接続された Windows Vista パソコンが不正ソフトウェアのダウンロードによってマルウェア感染し，ネットワーク内に存在する他の端末に対してポートスキャンが実行されている．

評価で用いたパラメータは，表 2 に示した通りである．T_{min}, T_{max}, C_{max} は，本評価で用いたデータセット向けに妥当と考えられる値を設定した．評価においては，観測期間 W を変化させた．イベント検出周期は 3 秒とした．W は 0˜ 3000(ミリ秒) の間で 200 ミリ秒おきに変化させた．それぞれのイベントについて，観測タイミングを変化させ，100 回の試行を行った．*response* の評価結果の値は全試行の平均値を表している．

4.1.3　RA の性能

図 4 は，前述のデータセットを用いた RA イベントの検出性能を示している．X 軸は観測期間 W である．本図および以降の図では，W に応じた結果が比較しやすいよう，*precision*, *recall*, *response* を同時に表示している．左 Y 軸は *precision*, *recall* を，右 Y 軸は *response*（秒）を表している．

W が小さい範囲では，T_{min} が，異常ではない状態，あるいは，異常であるにもかかわらず T_{min} に達しない状態が起きており，*precision*, *recall* ともに低下が見られる．また，応答時間も長くなっている．この傾向は W が大きくなるにつれて改善しており，$W = 1400$ のとき，$precision = 0.98$, $recall = 1.0$, $response = 2.1$ (秒) となり，$W > 1400$ ではほとんど同じ結果となった．すなわち，本評価のパラメータでは，$W \geq 1400$ であれば十分な性能を得ることができた．

4.1.4　AA の性能

図 5 は，AA-Single イベントの検出性能を示している．AA イベントは T_{max} を超える状態となった時点で発生する．AA-Single は非常に短い時間に突発的に発生する事象であるため，W が小さい時，観測漏れが生じ，*recall* が低下している．$recall = 1.0$ となるのは，$W = 3000$，すなわちイベント検出周期すべての期間観測を行った場合であった．*precision* は常に 1.0 であり，誤ったイベント検出は生じていない．また，観測されるまでにかかった応答時間は短く，$W = 3000$ のとき，$response = 1.54$(秒) であった．

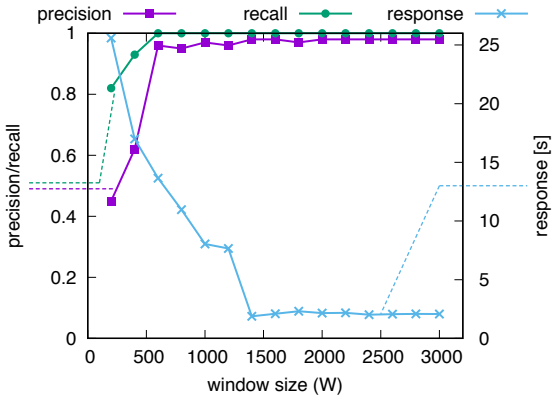


図 4　RA の検出性能

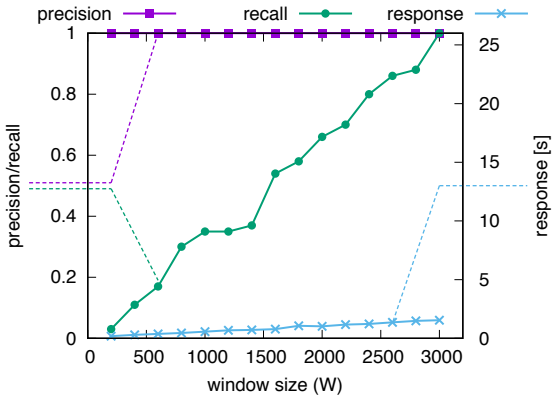


図 5　AA-Single の検出性能

図 6 は，AA-Multi イベントの検出性能を示している．傾向は, AA-Single と同様であり，W が小さい時に *recall* が低い．ただし，AA-Multi イベントは継続期間が長いため，観測漏れは少なく抑えられている．一方，*precision* は，$1200 \leq W \leq 2600$ のとき，1.0 であるが，$2800 \leq W$ で小さく低下し，$W = 3000$ のとき，$precision = 0.97$ となっている．これは，パラメータとした $T_{min} = 100$ では，AA-Multi イベントではない状態を誤って AA-Multi イベント認知してしまったことを表している．*response* は 1.5 秒から 2 秒の値となっており，$W = 1200$ の時，$response = 1.97$(秒) であった．

4.1.5　SA の検出性能

図 7 は，SA イベントの検出性能である[*1]．*precision*, *response* は W が大きくなると低下している．$W > 1000$ のときの *response* はほぼ 3 秒となった．一方，$W < 1000$ では非常に応答時間が長くなっている．これは，観測時間が短いと SA イベントの発生を見逃す確率が高まり，実際には発生している SA イベントの検出漏れが生じている

[*1] 見やすさを考慮して応答時間の表記はこの図のみ log scale となっている．

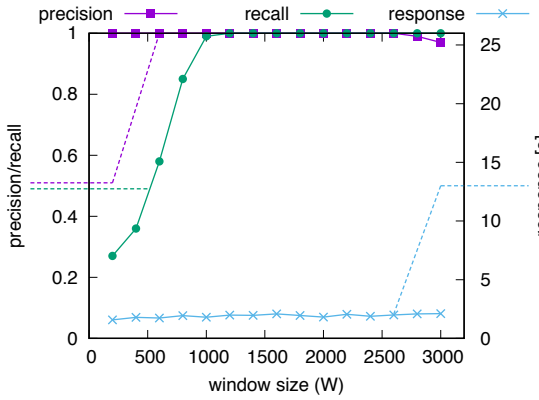


図 6　AA-Multi の検出性能

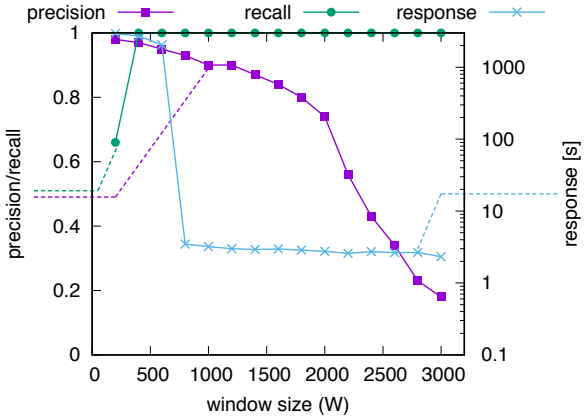


図 7　SA の性能

ためである．比較的バランスの良い結果が得られたのは，$W = 1200$,$W = 1400$ のときであり，$W = 1200$ のとき $response = 2.99$(秒)，$precision = 0.90$ となり，$W = 1400$ のとき $response = 2.91$(秒)，$precision = 0.87$ となった．いずれも，$recall = 1.00$ である．

評価に用いたデータセットでは，W が大きくなるにつれて $precision$ が低下している，これは観測時間が長いと，人が通常のネットワーク利用を行う動作において C_{max} を超える確率が高まり，FP_p が増えるためである．

4.2　実機上での性能

提案システムの機能を実装した Catalyst 9300 は，パケットスイッチ機器であり，異常イベント検知の機能を本来持っていない．したがって，本機能がスイッチ機器本来の機能であるパケット転送の性能にできるだけ影響を及ぼさない必要がある．

上記を確認するため，検出性能の評価で用いた CICIDS2017 [13] データセットを用いて，提案システムが実機に与える影響を検証した．本データセットは約 120 万個のパケットを含み, 約 19 万個の Flow から成る．評価にあたっては，パケット生成ツール (TCPreplay [14]) に対して上記データセットを入力してトラヒックを生成し，本機

表 3　実機上での性能

	Real モード (CPU 負荷)	500Mbps モード (CPU 負荷)	500Mbps モード (転送速度)
提案手法なし	1%	5%	363.26 Mbps
提案手法あり	4%	22%	363.91 Mbps

能を実装した Catalyst 9300 に入力して，レイヤ 2 接続した端末間のデータ転送を行った．パケット生成ツールが出力するトラヒックの帯域は設定可能であり，本評価では，タイムスタンプにしたがって生成する設定 (Real モード)，および 500Mbps で生成する設定 (500Mbps モード) を用いた．また，公平性を保つため，各計測の前にスイッチの再起動を行った．

本検証では，提案手法の有無に対する最大 CPU 負荷，実効転送速度を計測した．表 3 は，実行結果をまとめたものである．Real モードでは，提案手法なしの場合，CPU 負荷は 1 % であるのに対し，提案手法ありの場合は 4 % となった．また，500 Mbps モードでは，提案手法なしの場合 5 %，提案手法ありの場合は 22 % となった．それぞれ，トラヒック量に応じて CPU 負荷が上昇しており，500Mbps モードは Real モードと比べておよそ 5 倍程度の負荷がかかっていることがわかる．一方，実効転送速度は，いずれも 500Mbps モードにおいて，およそ 363Mbps から 364Mbps となっており，提案手法の有無によってほとんど差はなく，計測誤差の範囲に収まっている．

以上より，提案システムによってトラヒックに応じた CPU 負荷上昇は見られるものの，パケット転送性能への影響はほとんどないことがわかった．

5.　おわりに

ビッグデータの利活用を促進するため，プライバシ情報が多く扱われる IoT ネットワークではデータ漏洩などの脅威を軽減することの重要性が高まっている．とくに IoT ネットワークで重要な物理的な攻撃に対しては，迅速な発見と対応が要求される．従来の SNMP を用いたネットワーク機器の監視では，30 秒程度の問い合わせ間隔でインタフェース状態の異常検出が行われてきており，物理攻撃に対する対応速度として十分とは言えず，また，重要な異常を検出できない問題があった．評価で示した通り，提案システムではネットワーク性能を低下させることなく 3 秒以下の応答時間が実現できる．すなわち，物理攻撃に対応する異常イベントへの対応速度を 1/10 に短縮することが可能である．提案システムは，近年一般的になりつつある on-box programming が可能なネットワーク機器を用いて実装可能であり，IoT ネットワーク向けにコスト・性能・安全性のバランスが取れた対策となり得ると考えている．

一方，提案システムで現在用いているイベント検出方法

は比較的シンプルであり，評価で示した通り，適切なパラメータの設定が重要である．特に人がネットワークを利用する状況においては，適切に異常を検知するパラメータ設定は難しい．より精度高い異常検出の方法については，数多くの検討がなされている．代表的なものとしては，トラヒックの周期性に着目した方法 [15]，エントロピーに基づく検出方法 [16]，異常状態の機械学習に基づく手法 [17] などがある．このような高度な判定方法を適用することで，人が利用するネットワークにおいても，異常イベントの検出性能は向上可能と考えられる．一方，これらの判定方法は，実装が複雑となり，リソースを多く消費するため，実行が可能か，どの程度の応答性能が得られるか等の評価が必要となる．より高度な判定方法の適用は，今後の課題である．また，異常イベントそれぞれの発生頻度に応じてバッファサイズやパラメータを最適化するといった実装の効率化方法，ならびにその評価についても今後取り組んでいく．

参考文献

[1] J. D. Case, M. Fedor, M. L. Schoffstall, J. Davin, “RFC 1157: Simple network management protocol (SNMP),” IETF, 1990.
[2] P. Kasinathan, G. Costamagna, H. Khaleel, C. Pastrone, M.A.Spirito, “An IDS framework for internet of things empowered by 6LoWPAN,” In Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security, pp. 1337-1340, 2013.
[3] J. Quittek, T. Zseby, B. Claise, S. Zander, “RFC 3917: Requirements for IP flow information export (IPFIX),” IETF, 2004.
[4] B. Claise, “RFC 3954: Cisco systems netflow services export version 9,” IETF, 2004.
[5] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, A. Lakhina, “Impact of packet sampling on anomaly detection metrics,” In Proc. of the 6th ACM SIGCOMM conference on Internet measurement, pp. 159-164, 2006.
[6] R. Hofstede, V. Bartos, A. Sperotto, A. Pras, “Towards real-time intrusion detection for NetFlow and IPFIX. In Proc. of 2013 9th IEEE International Conference on Network and Service Management (CNSM), pp. 227-234, 2013.
[7] Linux Containers, `https://linuxcontainers.org`, (Online, last accessed: Jan. 30, 2019).
[8] V. Bollapragada, C. Murphy, R. White, “Inside Cisco IOS software architecture,” Cisco Press, 2000.
[9] Cisco NX-OS, `https://www.cisco.com/c/en/us/products/ios-nx-os-software/nx-os/index.html`, (Online, last accessed: Jan. 30, 2019).
[10] Cisco IOS Embedded Event Manager (EEM), `http://www.cisco.com/go/eem`, (Online, last accessed: Jan. 30, 2019).
[11] R. Presuhn, “RFC 3418: Management Information Base (MIB) for the Simple Network Management Protocol (SNMP),” IETF, 2002.
[12] E. Rosen, Y. Rekhter, “RFC 4364: BGP/MPLS IP Virtual Private Networks (VPNs),” IETF, 2006.
[13] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani, “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization,” In Proc. of the 4th International Conference on Information Systems Security and Privacy (ICISSP), pp. 108-116, 2018.
[14] TCPReplay, `http://tcpreplay.appneta.com`,(Online, last accessed: 28 Dec 2018).
[15] T. Akgul, S. Baykut, M. Erol-Kantarci, S. Oktug, “Periodicity-based anomalies in self-similar network traffic flow measurements,” IEEE Transactions on Instrumentation and Measurement, 60 (4), pp. 1358-1366, 2011.
[16] G. Nychis, V. Sekar, D. G. Andersen, H. Kim, H. Zhang, “An empirical evaluation of entropy-based traffic anomaly detection,” In Proc. of the 8th ACM SIGCOMM conference on Internet measurement, pp. 151-156, 2008.
[17] T. Kimura, A. Watanabe, T. Toyono, K. Ishibashi, “Proactive failure detection learning generation patterns of large-scale network logs,” IEICE Transactions on Communications, 2018EBP3103, 2018.