ネットワーク攻撃に対する端末非依存型検知方式のOpenFlowコントローラ上への実装と評価

宮崎, 亮輔
九州大学

川本, 淳平
九州大学

松本, 晋一
九州先端科学技術研究所

櫻井, 幸一
九州大学

出版情報：マルチメディア，分散，協調とモバイルシンポジウム．2015，pp.918-926，2015-07-08
バージョン：
権利関係：

ネットワーク攻撃に対する端末非依存型検知方式のOpenFlowコントローラ上への実装と評価

宮崎 亮輔[1]　川本 淳平[1]　松本 晋一[2]　櫻井 幸一[1]

概要：本稿では，ネットワーク仮想化技術である Software-Defined Network (SDN) において，マルチエージェントを用いてサイバー攻撃を検知する手法を提案する．我々がこれまでに提案してきた手法では，SDN 内に単一のエージェントを配置し攻撃検知を行っていた．しかし，単一エージェントでは検知結果にノイズが含まれることが問題であった．本稿では，監視するネットワークデータに関する多様性，異常と判断する閾値に関する多様性並びにエージェントの寿命に関する多様性を導入し，単一エージェントによる過度な検出を抑え複数の異常傾向を考慮した検知を提案する．また，比較評価実験により，多様性の導入によって検知結果ノイズが削減できたことを確認した．

Implementation and Evaluation of Host Independent Network Detection System on OpenFlow Controller

RYOSUKE MIYAZAKI[1]　JUNPEI KAWAMOTO[1]　SHINICHI MATSUMOTO[2]　KOICHI SAKURAI[1]

1. はじめに

近年，コンピュータネットワークは急速に普及してきている．それに伴い，以前は愉快犯によるサイバー攻撃が主であったのが，近年では愉快犯に加えて金銭や個人情報を狙ったサイバー攻撃へと，攻撃形態が多様化してきている．その一方で，ウイルス対策ソフト等を積極的に導入しないような，ネットワークセキュリティに対する意識が低いユーザが存在する．そういったユーザに対しても有効なセキュリティを提供することが必要である．特に，図 1 に示す様に，コンピュータに侵入し，不正な行動を行うプログラムであるマルウェアの種類は年々増加しており，その被害は大きな社会問題となっている．

更に，近年ではネットワークセキュリティはパーソナルコンピュータに限らない状況となっている．ハードウェア性能の向上により，ネットワークに接続する機能を持った組み込み機器が急速に普及してきている．しかし，これらの機器に対するセキュリティが不十分なまま運用されていることが多く，その脆弱性をついた攻撃も見られるようになっており，組み込み機器に対するセキュリティも問題の一つとなっている．

マルウェアから自身を守る手段として，従来様々な手法が確立されてきたが [11]，その中でもパターンマッチング手法は広く使用されている．パターンマッチング手法とは，既知のマルウェア群からなるデータベースを参照し，マルウェアを検出する手法である．しかし，上述した様にマルウェアは日々進化・増加しており，未知のものに対しては検知することが難しい．一方，近年注目され始めた手法として，Moving Target Defense という防衛概念 [5][6] が存在する．MTD とは，より非決定的で，より非静的で，より不均質となるようにシステムの攻撃面 [8] を変化させる防衛概念であり [1][7]，米国大統領率いる行政機関の National Science and Technology Council (NSTC) で提唱・推進されている．しかし，MTD は常にユーザの環境が変化するので，正当なユーザにまで影響を及ぼしてしまうという問題が存在する．

これらの二つの検知・防衛手法の問題を解決した既存手法として，Ant-Based Cyber Defense(ABCD)[2][3][4] が存

[1] 九州大学
Kyushu University
[2] 九州先端科学技術研究所
Institute of Systems, Information Technologies and Nanotechnologies (ISIT)

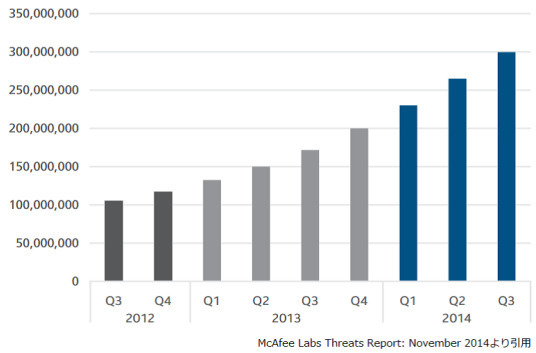

図 1　マルウェア数の遷移 [9]

在する．しかし，この手法では，防衛対象の各端末にソフトウェアの導入を必須としている．したがって，上述したネットワークに接続する機能を持った組み込み機器に対して適用することが出来ないという問題が存在する．

そこで，我々は MTD を用いた攻撃検知を，機器を改変することなく導入する手法 [13] を提案した．しかし，この手法では端末の異常判定に用いる閾値やパラメータが固定されており，MTD の定義の一つである多様性を満たしていない．即ち，固定された閾値が攻撃者に知られることで，容易に検知を回避されてしまう可能性があるという問題が存在する．

本研究では，我々が提案した既存手法 [13] に多様性を持たせることで，MTD の特性を持つように改善する手法を提案する．異常判定の閾値は毎回ランダムに決定し，パラメータの種類は毎回ランダムに選択する．既存手法と比較して，より幅広い通信に対して検知することが可能となる．実験では，SDN のプロトコルとして OpenFlow を用い，提案手法を実現する OpenFlow コントローラを作成した．

2. Software-Defined Network(SDN)

Software-Defined Network とは，ネットワークにおける構成や機能をソフトウェアの操作のみで動的に設定，変更できるネットワークのことを指す．

OpenFlow

OpenFlow は SDN を実現するためのプロトコルの 1 つである．これを用いることで従来ハードウェアで静的であったネットワークを，ソフトウェアで自由に記述することができるので，最終的にはネットワーク全体をプログラムによって制御することが可能となる．

OpenFlow は従来のネットワーク機器が担っていた経路制御機能とデータ転送機能を分離している．経路制御は OpenFlow コントローラ，データ転送は OpenFlow スイッチと呼ばれるものがそれぞれを担い，コントローラとスイッチは OpenFlow プロトコルで通信を行っている（図 2）．一般的にはコントローラはソフトウェアで実装されるので，従来のネットワークにはない動的なネットワーク制御を行うことができる．

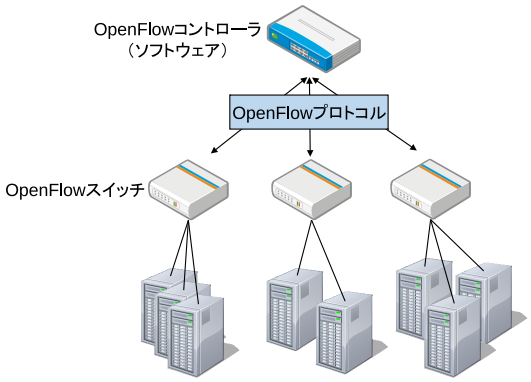

図 2　OpenFlow コントローラと OpenFlow スイッチ

各々のスイッチは，フローテーブルと呼ばれる，パケットの条件（マッチングルール）とその扱い（アクション）が紐付けられたリストを持っており，パケットが来る度にそのリストを参照して動作を決定する．例えば，「宛先 IP アドレスが X ならば，スイッチの Y 番ポートから出力する」「TCP の Z 番ポートに入って来たパケットは破棄する」といったものである．マッチングルールに合致しないパケットが来た場合は，コントローラに問い合わせて指示を仰ぐ．コントローラから受けた指示内容は，新たなマッチングルール及びアクションとして，フローテーブルに書き込まれ更新される．

2.1 OpenFlow の仕組み

(1) スイッチとコントローラ間の接続の確立

まず始めにスイッチとコントローラは接続を確立する．これをセキュアチャネルと呼び，通常では TCP プロトコルを用いてスイッチからコントローラへ接続を行う．スイッチ・コントローラの双方が対応していれば，より安全な TLS プロトコルを使用することもある．

(2) 未知パケットの受信

次に，OpenFlow プロトコルのバージョンを確認し，スイッチとコントローラは互いに自分のバージョン番号を載せたメッセージを送り合う．相手と同じバージョンのメッセージを送ることができれば成功となり，次のステップへ進む．

(3) フローテーブルの更新とパケットの転送

スイッチから Packet In メッセージを受け取ったコントローラは，事前に記述されたプログラムに従ってそのパケットの動作を決定し，その内容をフローエントリとしてスイッチに送信する．スイッチは，自身のフローテーブルにその時のパケット内容（マッチングルール）と動作（アクション）を書き込み更新する．また，コントローラは Packet In を起こしたパケット本体を正しい送り先に流す．これを Packet Out メッ

セージと呼ぶ．

次にスイッチにフローエントリと一致するパケットが来た場合は，既にその場合の動作はスイッチの持つフローテーブルに書かれているので，コントローラに問い合わせることなく，スイッチのみでパケットの処理を行うことができる．

2.2 フローエントリ

フローエントリは以下の 3 要素から成っている．

- マッチングルール
- アクション
- 統計情報

(1) マッチングルール

マッチングルールは，スイッチが未知のパケットを受け取った場合にどの様な動作を行うとかを決める条件である．例えば，「ARP プロトコルであった場合は」「TCP の受信ポートが 25 番であった場合は」という条件に合致した時に，スイッチは特定の動作を行うことになる．OpenFlow1.0 では，表 1 に示す 12 種類の条件が規定されており [14]，マッチングルールで指定できる条件を自由に組み合わせて通信を制御することができる．

(2) アクション

アクションは，入ってきたパケットをどう処理するかという部分である．実際のルータでは，パケットの MAC アドレス部分を書き換えてルーティングを行っているが，このパケットの書き換えも OpenFlow のアクションの一つである．アクションには大きく分けて 4 種類存在し，パケットを指定したポートから出す Forward，パケットの中身を書き換える Modify-Field，パケットを破棄する Drop，ポート毎に指定されたスイッチのキューに入れる Enqueue から成る．

(3) 統計情報

OpenFlow ではフローエントリ毎に，以下の統計情報を取得することができる．

- 受信パケット数
- 受信バイト数
- フローエントリが作られてからの経過時間

3. 先行研究

この章では MTD を用いた先行研究について述べる．

Jereme らは，マルチエージェントシステムを用いてエージェントの密度により攻撃を検知しており [2][3]，各々の端末に対してソフトウェアを導入している．端末を監視するソフトウェアと，各々で異なるパラメータを持つセンサーアントの 2 つを組み合わせることで，端末に攻撃があった際に，その端末にセンサーアントが集まるシステムとなっている．この手法が見る値は CPU 使用率やメモリ使用率

表 1 マッチングルールで指定できる 12 種類の条件

名前	説明
Ingress Port	スイッチの物理ポート番号
Ether src	送信元 MAC アドレス
Ether dst	宛先 MAC アドレス
Ether type	イーサネットの種別
IP src	送信元 IP アドレス
IP dst	宛先 IP アドレス
IP proto	IP のプロトコル種別
IP ToS bits	IP の ToS 情報
TCP/UDP src port	TCP/UDP の送信元ポート番号
TCP/UDP dst port	TCP/UDP の宛先ポート番号
VLAN id	VLAN ID
VLAN priority	VLAN PCP の値 (CoS)

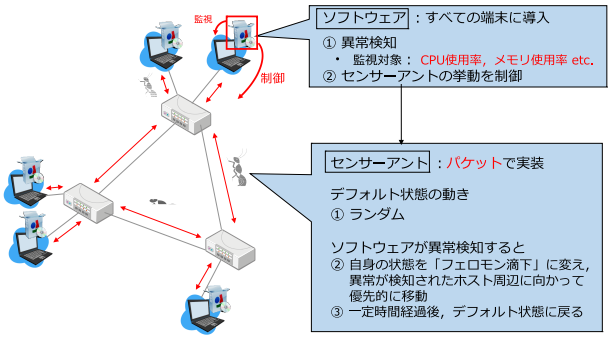

図 3 Ant-Based Cyber Security の概念図

等の記述的な値であるので，未知の攻撃に対しても検知できるという点において優れている．しかし，この手法は各端末に対してソフトウェアを導入することが必要であり，組み込み機器の様な，リソースの限られた端末に対しては導入することが難しいという問題がある．

そこで，我々はその問題点を解決するために，Jereme らの手法と SDN を組み合わせる手法を提案した [13]．SDN として OpenFlow を用いることで，各端末に対してソフトウェアを導入する必要がなくなり，組み込み端末の様なリソースの限られた環境に対しても Jereme らの手法を導入することが可能となった．しかし，我々の既存手法では，端末の異常判定に用いる閾値やパラメータが固定されており，MTD の定義の一つである多様性を満たしていない．即ち，固定された閾値が攻撃者に知られることで，容易に検知を回避されてしまう可能性があるという問題がある．

4. 提案手法

本論文では，既存手法 [13] に多様性を加え MTD として成立させることで，より幅広い状況において運用可能なマルウェア対策技術を提案する．SDN には OpenFlow1.0 を用い，実験のために端末へ導入するソフトウェアの役割を代替する OpenFlow コントローラを作成した．具体的には，

表 2　自然界のアリと提案手法における各要素の対応関係

自然界のアリ	提案手法
巣	対象ネットワークの各スイッチ
エサ	攻撃されたホストが接続されたスイッチ
アリ	パケット

表 3　OpenFlow スイッチの持つパラメータ

パラメータ名	説明
:ant_count	現在そのスイッチに留まっているアントパケットの数
:pheromone_count(x)	スイッチ間のフェロモンの値

- アリに見立てたパケットに異なる 2 つの状態を持たせ，パケットの状態に応じて異なるルーティングルールを適用させる
- 常に端末の通信を監視しており，通信挙動に応じてアリの状態を書き換え，異なるルーティングルールを適用させる

以上の要件を満たす OpenFlow コントローラを作成した．なお，上記のルーティングルールを適用させるのは本手法で使用したアリに見立てたパケットのみであり，それ以外のパケットは影響を受けない．

4.1　自然界のアリの動き

自然界のアリは，以下の規則に従ってエサを効率的に巣まで運ぶことが知られている [*1]．

- エサを持っていない時は，周囲のフェロモンの濃度が高い方向に向かって歩く．ただし，フェロモンが全くない時は自由に動く．エサのある場所に到着したらエサを持つ．
- エサを持っている時は，その場にフェロモンを落としながら，巣の方向に向かって歩く．巣に戻ったらエサを巣に置いて，再び歩き出す．

なお，フェロモンは時間の経過と共に蒸発していく．以上のアリの動きを OpenFlow コントローラで実装した．本提案手法では，自然界のアリのエサ取り行動を，表 2 に示す様に置き換えて実装している．また，フェロモンはスイッチ間の経路毎に存在し，時間と共に蒸発係数を掛けることによって蒸発を実装している．

4.2　OpenFlow スイッチについて

本手法で使用する OpenFlow スイッチは，自然界のアリでは巣を表しており，表 3 に示すパラメータを保持している．

- :ant_count
  本手法では各々のスイッチに集まるアントパケットの数によって攻撃の検知を行うので，各スイッチは自身を訪れたアントパケットの数を保持している．
- :pheromone_count(x)
  各々のスイッチは，自身と繋がっている他のスイッチ間の経路毎に後述するフェロモンの値を保持している（図 4）．ここで，引数 x は接続先のスイッチの IP アドレス，MAC アドレスの組である．

[*1] アリのフェロモンを使ったえさ取り行動 http://www.alife.cs.is.nagoya-u.ac.jp/~reiji/aw/ants.html [Accessed 10 2 2015]

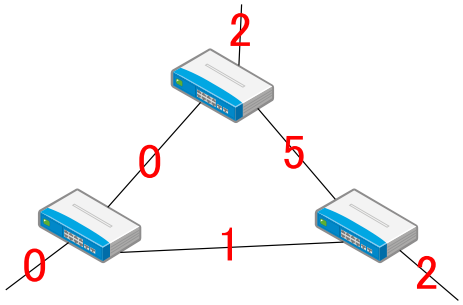

図 4　フェロモンカウント

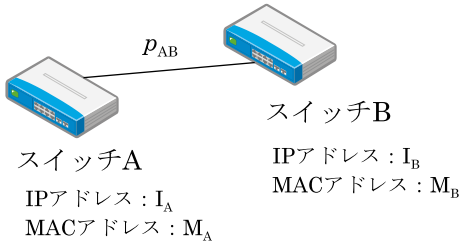

図 5　スイッチ A-B

自然界のアリは，エサを発見するとフェロモンを滴下しながら巣まで戻っていく．他のアリはこのフェロモンの跡を辿ってエサまで進み，最終的にエサから巣までの最短経路上に行列を作る．本手法では，攻撃されたホストが接続されたスイッチをエサに見立て，このスイッチを経由したアントパケットはフェロモンを滴下しながら巣（そのアントパケットが生起したスイッチ）まで戻る．

ここで，図 5 の様にスイッチ A とスイッチ B が互いに接続されているとする．今，アントパケットがスイッチ A からスイッチ B に転送されたとする．この時，スイッチ A は自身の持つ変数 pheromone_count($[I_B, M_B]$) を 1 増やし，スイッチ B も自身の持つ変数 pheromone_count($[I_A, M_A]$) を 1 増やす．この時，スイッチ AB 間の経路のフェロモン値 p_{AB} を以下の式で定める．

$$p_{AB} = \text{pheromone\_count}([I_A, M_A]) + \text{pheromone\_count}([I_B, M_B])$$

アントパケットは，通常スイッチ周辺の経路の内，フェロモンカウントが最大であるような経路を通って次のスイッチへ転送される．また，このフェロモンカウントは時間と共に蒸発するようにしてあり，その蒸発係数を e とする．e は 5[秒] おきにかけられる．今，:pheromone_count= 1 とし，この値が T[秒] 後に少なくとも ε まで残っていること

表 4　アントパケットの持つパラメータ

パラメータ名	説明
:ant	アントの状態を表す． 0: デフォルト 1: フェロモン滴下中
:nest_IP	パケットのソース IP アドレス
:packet_count	閾値（パケット数）
:packet_size	閾値（パケットサイズ）
:life_count	アントの余命

を保証する為には，$e > (\varepsilon)^{\frac{5}{T}}$ であればよい．:ant_count の扱いについては，4.4.3 節にて後述する．

4.3　アントパケット

本手法では，TCP のペイロード部分に表 4 に示す情報を持たせた IP パケットを，各々のスイッチから一定間隔で発している．このパケットは自然界のアリを模しており，図 6 に示す動きを繰り返している．ただし，各々のアントパケットは寿命を持っており，1 回ホップする毎に齢を取っていき，異常の報告を受けないまま 10 歳に達した時点で消滅する．

- :ant

  自然界のアリは，エサを探している通常の状態と，エサを発見した後にフェロモンを滴下しながら巣まで戻る状態の 2 種類の状態がある．:ant パラメータの 0 と 1 はこの 2 つの状態を表しており，0 がデフォルト値，1 がフェロモン滴下中を表している．
- :nest_IP

  自然界のアリは，エサを発見した後はフェロモンを滴下しながら巣まで戻る習性がある．本手法で使用するアントパケットも，異常のあるホストが接続されたスイッチを発見した際に，そのアントパケットが生起したスイッチまで戻る．その為に巣の場所，即ちアントパケットのソース IP アドレスを保持している．
- :packet_count 及び:packet_size

  アントパケットは生まれた時点でランダムに閾値が与えられる．この値は 4.4.3 節で示す異常判定の際に使用される．
- :life_count

  もしアントパケットに寿命が無ければ，異常が報告されない限りネットワーク上のアントパケットの数は際限なく増加していってしまい，時間と共に OpenFlow コントローラに対する負荷が増加する．従って，寿命は短い方が望ましい．一方で，寿命を短くし過ぎるとアントパケットが有効な範囲が限られてしまい，ネットワークの規模によっては検知が困難になる．

各々のパラメータの扱いは，4.4.3 節にて後述する．

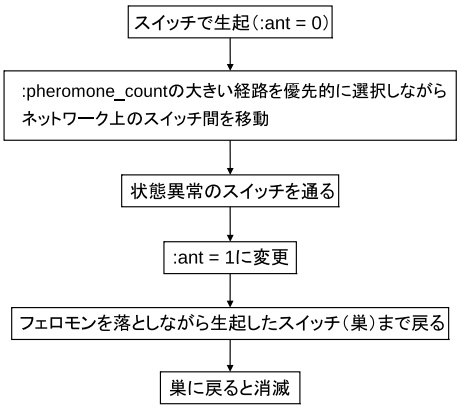

図 6　アントパケットのライフサイクル

4.4　OpenFlow コントローラ

本手法の為に作成した OpenFlow コントローラ（以下，コントローラ）について説明する．

4.4.1　基本機能

作成したコントローラは以下の機能を有している．

- スイッチに接続されているホストのトラフィックの監視・解析
- アントパケットの制御

それぞれの機能に関して以下に詳しく述べる．

4.4.2　スイッチに接続されているホストのトラフィックの監視・解析

各スイッチに接続されているホストのトラフィックをコントローラで常に監視している．監視する対象は以下の通りである．

- 1 秒あたりのパケット数
- 1 秒あたりの通信量

コントローラは，各スイッチからポート毎のパケット数及び通信量（単位：バイト）の報告を定期的に受けており，その値を 60 秒毎に区切り，その値の差分を 80 秒毎に計算し，1 秒毎の上記の値を各々算出している．これらの値を監視し，予め設定した閾値もしくはアントパケットが保持している閾値を越えると自状態を「異常」とする．この状態が「正常」か「異常」かで，アントパケットの送り方が異なる．その送り方については，次の 4.4.3 節で示す．

4.4.3　アントパケットの制御

既存手法 [2][3] では，アントパケットの制御アルゴリズムが公開されていない為，自然界のアリの動き [*2] を参考に，アントパケット制御のアルゴリズムを以下の様に設計した．各スイッチに来たアントパケットは，図 7 に示すフローチャートに従って制御される．ここで，図中の巣とはアントパケットが生起したスイッチを指し，異常発見とは上の小節で示したトラフィック解析によるホストの監視結果を指す．図中の p は:pheromone_count の値を表してお

[*2] アリのフェロモンを使ったえさ取り行動 http://www.alife.cs.is.nagoya-u.ac.jp/~reiji/aw/ants.html [Accessed 10 2 2015]

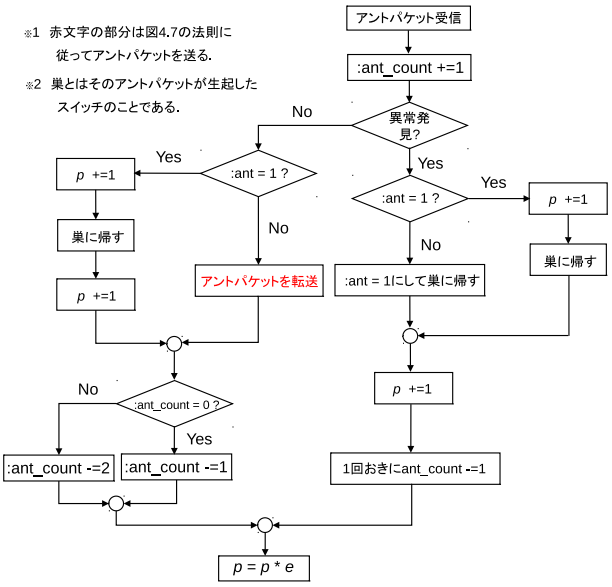

図 7　アントパケットの制御フローチャート

り，$p = p \times e$ は，上で述べたフェロモンカウントの蒸発を表している．また，赤色で示した「アントパケットを転送」の部分は，次の小々節で示す方法に従って次の送り先を決めている．

まず，スイッチはアントパケットを受け取ると，自身の持つパラメータである:ant_count を 1 増やす．次に，スイッチは上述したトラッフィク解析による単位時間あたりのパケット数・通信バイト数を見て，事前に設定した閾値もしくはアントパケットが保持している閾値を超えていれば異常，超えていなければ正常と判断し，次へ移る．

(1) スイッチの状態が異常のとき

(a) :ant= 0 のとき

アントパケットの状態，即ち:ant の値を 1 にした後，アントの持つパラメータ:nest_IP を目指して転送する．

(b) :ant= 1 のとき

アントパケットを受信したポートの経路の:pheromone_count を 1 増やす．その後，そのアントパケットを巣に帰す，即ち:nest_IP を目指して転送する．

その後，スイッチの:pheromone_count を 1 増やし，異常のあるスイッチにおいてアントパケットの数が増加するように，:ant_count の値を 1 回おきに 1 減らしている．

(2) スイッチの状態が正常のとき

(a) :ant= 0 のとき

アントパケットのパラメータは一切変えずに，図 8 の法則に従って決定した次の転送先に送る．

(b) :ant= 1 のとき

アントパケットを受信したポートの経路の:pheromone_count を 1 増やす．その後，そのアントパケットを巣に帰す，即ち:nest_IP を目指して転送する．その後，スイッチの:pheromone_count の値を 1 増やす．

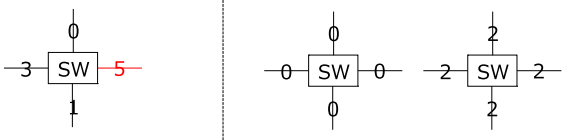

図 8　通常時の「0」アントパケットの制御

さらにその後，アントパケットを受け取ったスイッチの:ant_count の値によって:ant_count の減らし方が異なる．

(a) :ant_count= 0 のとき

:ant_count の値が変化しないように 1 減らす．

(b) :ant_count≠ 0 のとき

状態が正常であるのに:ant_count が 0 でない場合は，アントパケットを解散させるために:ant_count を 1 つ余分に減らす．

その後，全てのスイッチの:pheromone_count の値にフェロモン蒸発係数 $e(e < 1)$ をかける．以上が，スイッチがアントパケットを処理する一連の流れである．

4.4.4　スイッチの状態が正常かつ:ant=0 の時のアントパケットの送り方

通常，アントパケットは「0」状態であり，このパケットが正常状態のスイッチに来た場合は，周囲の:pheromone_count 値の内，最大のものの中からランダムに行き先を決定する（図 8）．

4.5　多様性

MTD とは，より非決定的で，より非静的で，より不均質なシステムを指す [1][7]．既存手法を MTD として機能させるには，上記に倣って多様性を持たせることが必要である．既存手法において，異常判定にはパラメータの種類，閾値を全て固定していた．従って，事前に設定した閾値を超える状況にのみアントパケットは集合した．しかし，実際のネットワークでは多種多様な通信が行われ，単一の閾値だけでは異常と正常を判別することが困難である．即ち，閾値は動的に変化させる必要がある．パラメータの種類も同様に変化させることが必要である．以下に，既存手法が持ち得る多様性の要素を列挙する．

- 異常判定に用いるパラメータの種類の集合 P
- 異常判定の閾値 s_1, s_2
- アントパケットの寿命

ここで，異常判定に用いるパラメータの種類は単位時間あたりのパケット数 p_1，通信量 p_2 を指し，s_1, s_2 はそれぞれの閾値を指す．P に関しては p_1 のみ，p_2 のみ，p_1 か

つ p_2 の 3 種類が存在する．3 種類の中からランダムに一つを選択し，コントローラでの異常判定に使用する．また，s_1, s_2 に関しては，ある閾値 t_1, t_2 を事前に選択し，アントパケットが生起する度に $0 < s_1 < t_1, 0 < s_2 < t_2$ を満たすような s_1, s_2 を与える．この値の変化により，各々のアントパケットは異なる挙動を示し，幅広い反応性を持つので，多様性を実現させることが可能である．寿命に関しても，値を変化させることでネットワーク上のアントパケットの数が増減し，多様性を与える．これらの要素をランダムに変化させることで，既存手法に多様性を与えることが可能となる．

5. 実験

既存手法が持ち得る多様性を評価する実験を行った．まず，「閾値」「異常判定に用いるパラメータの種類」の二つの要素を固定もしくはランダムと変化させて最適な組み合わせを求める実験を行った．次に，その結果から得られた最適な組み合わせを用いて，各パラメータの値を変えることでその結果を評価した．実験は，CCCDATAset2008[12] によるボットネット通信データに関して評価した．

5.1 実験環境

本研究において実験に使用した環境は以下の通りである．

- OS: Ubuntu 14.04 LTS
- CPU: Intel Corei7-3960X
- Main Memory: 32GB

5.2 CCCDATAset を用いたボットネット検知実験

この小節では，実際に攻撃のあった際のトラフィック観測データである CCCDATAset2008[12] を用いたボットネット検知実験について述べる．CCCDATAset2008 のハニーポットに含まれる pcap ファイルの中で，2008 年 4 月 28 日 0 時 4 分 34 秒から 2008 年 4 月 28 日 1 時 15 分 36 秒のデータを切り出し，Tcpreplay を用いて実験用の仮想 PC のネットワークインターフェースから 4220 秒間再放流した．図 9 のホスト 8 をハニーポットの IP アドレスとし，実際にボットネットの行った通信を検知できるかどうかを実験した．また，ホスト 8 以外のホストは仮想ホストであり，互いに Ubuntu の network-monitor サービスによる通信を行っている．実験は表 5 に示す 4 種類の実験を行った．実験に用いた各パラメータ値を表 6 に示す．この時のアントパケットの生起間隔は 1 秒とした．また，攻撃通信データは実験開始から 10 秒後に放流を開始した．ここで，ハニーポットとはボットネット通信を集める為に設置する，囮用のコンピュータのことを指す．

更に，表 5 の中で，この実験で得られた最適な組み合わせ C を用い，アントパケットの寿命及び閾値を変化させた．この時に用いた各パラメータ値を表 7 に示す．

表 5　行った 4 種類の実験

		パラメータの種類	
		静的	ランダム
閾値	静的	A	B
	ランダム	C	D

表 6　ボットネット検知実験で用いた各パラメータの値

実験	e	状態異常閾値		判定条件
		パケット数 [パケット/秒]	通信量 [バイト/秒]	
A	0.98	50	500	or
B	0.98	0-50	500	or
C	0.98	50	0-500	ランダム
D	0.98	0-50	0-500	ランダム

表 7　寿命及び閾値の評価実験で用いた各パラメータの値

実験	e	状態異常閾値		:life_count
		パケット数 [パケット/秒]	通信量 [バイト/秒]	
E	0.98	10	100	100
F	0.98	50	500	100
G	0.98	10	100	10
H	0.98	50	500	10

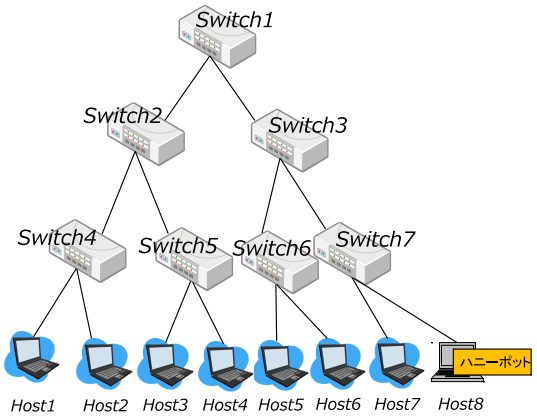

図 9　ボットネット検知実験で用いたネットワーク構成

5.3 実験結果と考察

ボットネット検知実験 A〜D の結果をそれぞれ図 10〜図 13 に示す．実験 A に関して，閾値を高い値で固定しているので，スイッチ 7 以外にはアントパケットがほぼ集まらなかった．また，閾値以下の時にはアントパケットが集まらないので，全体的なアントパケット数も実験 A〜D の中で最も少なくなった．実験 B に関して，閾値をランダムに設定したことで，微小な通信に対してもアリが集まっており，スイッチ 7 以外にも最大 20 程度のアントパケットが集まった．なお，4800 秒〜5000 秒においてアントパケットの減少が止まっているが，これはその時間帯において発生したノイズに対してアントパケットが反応し，一時

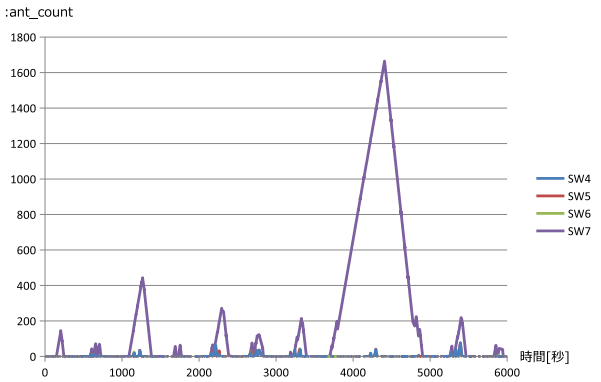

図 10　実験 A: パケット数 50，通信量 500 バイト

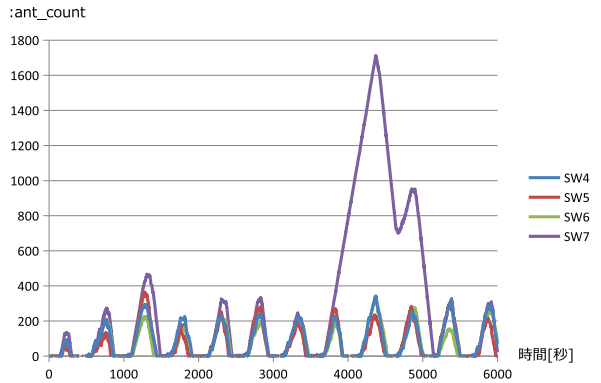

図 11　実験 B: パケット数 0-50，通信量 500 バイト

的に集まっていることが原因だと考えられる．実験 C 及び D に関しては，スイッチ 7 以外では最大でもアントパケットは 2 しか集まらず，ノイズの影響を受けることがほぼなくなった．また，C と D を比較すると C の方がノイズが少ない結果となった．

以上の実験結果より，閾値にランダム性を持たせた場合はノイズが増加し，パラメータの種類を変化させた場合はノイズが減少すると考えられる．従って，多様性を確保しつつノイズ耐性を持たせるためには，異常判定時のパラメータの種類をランダムに変化させれば良いと言える．

次に，実験 E〜H の結果をそれぞれ図 14〜図 17 に示す．いずれの結果も，同じ形になった．即ち，アントパケットの寿命を 100 に設定した場合と，10 に設定した場合において結果は変化しなかった．これは，実験のために作成した図 9 に示すネットワークにおいて，Host8 に対して行われたトラフィックがそれ以外の Host と比較してきわめて大きいことが原因だと考えられる．Host8 で観測されたトラフィックが大きすぎるので，少ないホップ数でアントパケットは容易に「1」状態へ移行してしまう．従って，アントパケットの寿命は高々 1 あれば十分という状況になっており，寿命による検知能力の評価が行うことが出来なかった．以上より，アントパケットの寿命を 100 から 10 に下げたことで，ネットワークの負荷は下がったが，それによる検知性能の変化は評価できなかった．

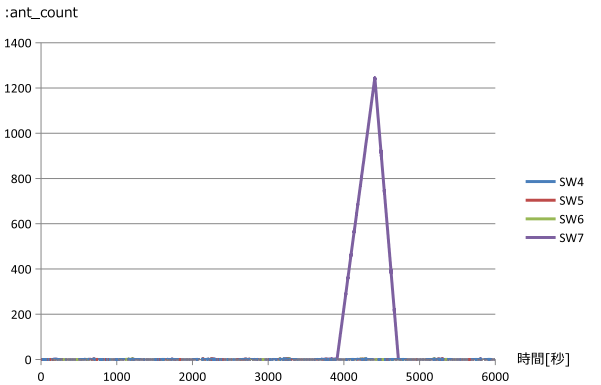

図 12　実験 C: パケット数 50，通信量 0-500 バイト

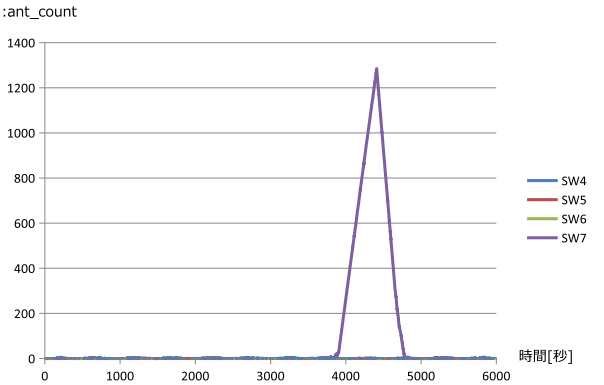

図 13　実験 D: パケット数 0-50，通信量 0-500 バイト

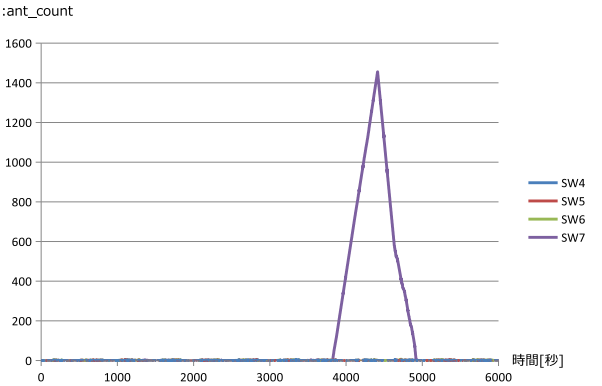

図 14　実験 E: パケット数 10，通信量 100 バイト，寿命 100

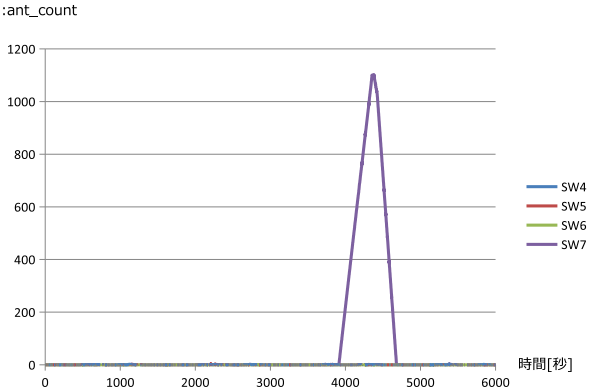

図 15　実験 F: パケット数 50，通信量 500 バイト，寿命 100

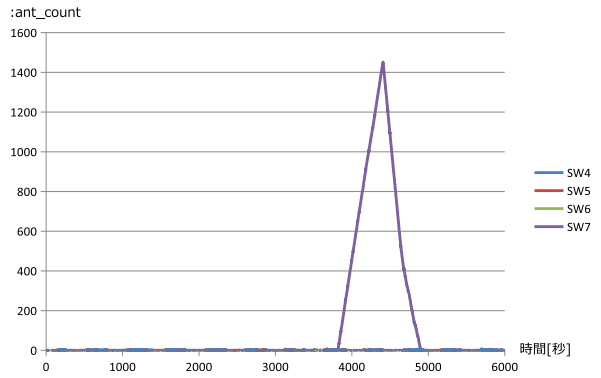

図 16 実験 G: パケット数 10，通信量 100 バイト，寿命 10

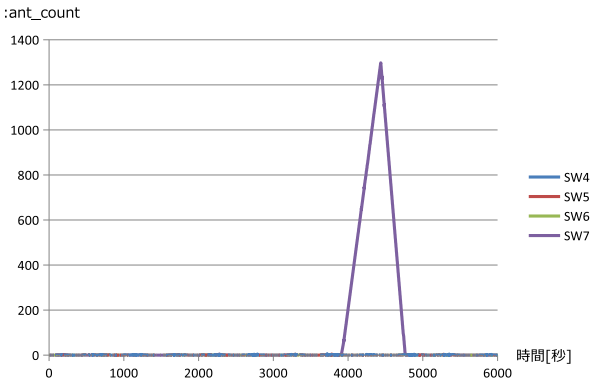

図 17 実験 H: パケット数 50，通信量 500 バイト，寿命 10

6. 結論

本提案手法を用いて，アントパケットに多様性を持たせることで，既存手法に多様性を持たせ Moving Target Defense の条件を満たすようにした．その際に，パラメータの種類をランダムに変化させることで，多様性とノイズ耐性を両立させることが可能であることを示した．しかし，本提案手法でランダム性を持たせたパラメータはパケット数と通信量のみに基づくことから，大小様々な通信を断続的に行っている様な環境においてはマルウェアの検知を行うことは難しい．従って，今後は更にアントパケットの制御に用いるパラメータの種類を増やし，より多様性を大きくすることが今後の課題である．また，今回の実験ではネットワークトラフィックの偏りが原因で，アントパケットの寿命による検知性能の評価を行うことができなかった．今後は，攻撃を受けることを想定したホスト以外のホストにも，通常時にキャプチャしたパケットデータを流すことで，トラフィックの偏りが生じないようにして実験を行う必要がある．

参考文献

[1] Executive Office of the President National Science and Technology Council, “Trustworthy Cyberspace: Strategic Plan for the Federal Cybersecurity Research and Development Program” pp.8-9, 2011.

[2] Jereme N. Haack, Glenn A. Fink, Wendy M. Maiden, A. David McKinnon, Steven J. Templeton, Errin W. Fulp, “Ant-Based Cyber Security” *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*. pp.918-926, 2011.

[3] Glenn A. Fink, Jereme N. Haack, A. David McKinnon, Errin W. Fulp, “Defense on the Move: Ant-Based Cyber Defense” *Security & Privacy, IEEE*. Vol. 12, Issue: 2, pp.36-43, 2014.

[4] Glenn A. Fink, A. David McKinnon, “Effects of Network Delays on Swarming in a Multi-agent Security System” *First International Workshop on Agents and CyberSecurity*. Article No. 11, 2014.

[5] Sushil Jajodia, Anup K. Ghosh, Vipin Swarup, Cliff Wang, X. Sean Wang, “Moving Target Defense” *Advances in Information Security*. Vol. 54, 2011.

[6] Jajodia, S., Ghosh, A.K., Subrahmanian, V.S., Swarup, V., Wang, C., Wang, X.S., “Moving Target Defense II” *Advances in Information Security*. Vol. 100, 2013.

[7] Thomas Hobson, Hamed Okhravi, David Bigelow, Robert Rudd, William Streilein, “On the Challenges of Effective Movement” *Proc. of the First ACM Workshop on Moving Target Defense*. pp.41-50, 2014.

[8] Pratyusa K. Manadhata, Jeannette M. Wing, “A Formal Model for a System’s Attack Surface” *Moving Target Defense*. pp.1-27, 2011.

[9] McAfee Labs, “McAfee Labs Threats Report: November 2014”. pp.29, 2014.

[10] Jean Tourrilhes, Puneet Sharma, Sujata Banerjee, Justin Pettit, “SDN and OpenFlow Evolution: A Standards Perspective” *Computer Software-Defined Networks*. pp.22-29, 2014.

[11] Manuel Egele, Theodoor Scholte, Engin Kirda, Christopher Kruegel , “A survey on automated dynamic malware-analysis techniques and tools” *ACM Computing Surveys (CSUR)*. Vol.44 Issue 2, 2012.

[12] 畑田充弘, 中津留勇, 寺田真敏, 篠田陽一, “マルウェア対策のための研究用データセットとワークショップを通じた研究成果の共有” 情報処理学会シンポジウムシリーズ, Vol.2009, No.11, CSS2009(MWS2009), pp.1-8, 2009.

[13] 宮崎 亮輔, 川本 淳平, 松本 晋一, 櫻井 幸一, “攻撃検知のための端末非依存型システムを実現する OpenFlow コントローラの実装と評価” 火の国情報シンポジウム, 2015.

[14] 高宮安仁, 鈴木一哉, 『クラウド時代のネットワーク技術 OpenFlow 実践入門』 技術評論社 pp.34, 2013.