攻撃検知の為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

宮崎 亮輔[1,2]　川本 淳平[1,2]　松本 晋一[1,2]　櫻井 幸一[1,2]

概要：本手法では，ネットワーク仮想化技術である Software-Defined Network(SDN) を用いてネットワークの監視を行い，端末に対する攻撃を検知する方式を実現した．提案手法では，スイッチ間でエージェントを送受信し，端末の通信状況に応じてエージェントを制御することで攻撃検知を実現している．SDN は各端末の通信を監視すると共に，エージェントの制御を行う．SDN が検知システムの制御を行うことにより，個々の端末への通信解析・攻撃検知用ソフトウェアの配備が不要となる．これにより，リソースや環境に依存しない汎用的な攻撃検知システムを実現することが可能となった．

Implementation and Evaluation of OpenFlow Controller for Host Independent Attack Detection System

Ryosuke Miyazaki[1,2]　Junpei Kawamoto[1,2]　Shinichi Matsumoto[1,2]　Koichi Saskurai[1,2]

Abstract: In this paper, we introduce a method of monitoring networks and detecting attacks for each host using Software-Defined Network (SDN), which is a network virtualization concept. In our method, each switch sends and receives agents to detect attacks by controlling agents based on hosts communication. SDN monitors each host communications and controls each agent at the same time. SDN's analysis of every connection makes attacks detection independent of host resources or environments, where SDN omits software introduction for network analysis and attacks detection procedure. Therefore, our method is a versatile system, which is independent of resources or environment.

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

[1] 九州大学, 福岡県福岡市西区元岡 744 番地
Kyushu University, 744, Motooka, Nishi-ku, Fukuoka, 819-0395, JAPAN
[2] 九州先端科学技術研究所, 福岡県福岡市早良区百道浜 2 丁目 1 番 22 号福岡 SRP センタービル 7 階
Institute of Systems, Information Technologies and Nanotechnologies(ISIT), Fukuoka SRP Center Building 7F, 2-1-22, Momochihama, Sawara-ku, Fukuoka, 814-0001, JAPAN

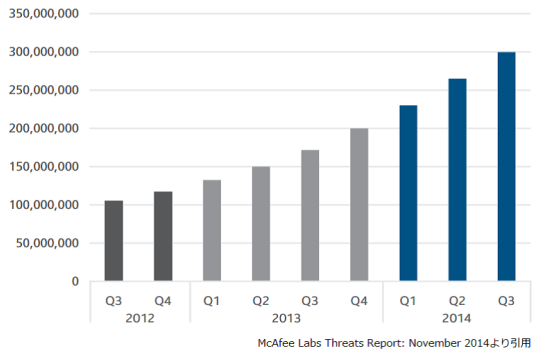

図 1 マルウェア数の遷移 [9]

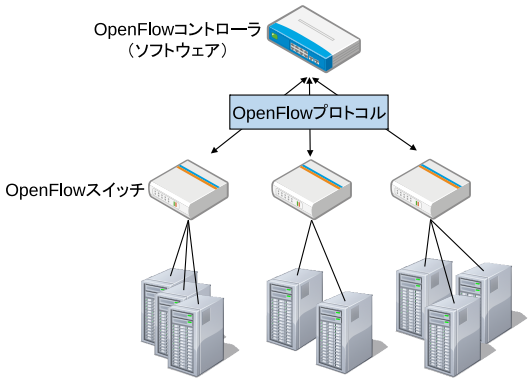

図 2 OpenFlow コントローラと OpenFlow スイッチ

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．

本研究では，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を提案する．ABCD ではマルチエージェントシステムを用いたのに対し，提案手法では，一種類のエージェントを使用するシングルエージェントシステムを用いている．ABCD と，ネットワークの仮想化技術である Software-Defined Network（SDN）[10] 及びシングルエージェントシステムを組み合わせる．SDN により，ネットワークの柔軟な監視を行うことが可能となり，従来機器側にインストールされたソフトウェアが担っていた機能を SDN が代替する．組み込み機器の通信に注目することで，より汎用的な検知システムを実現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．

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 [13]，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．

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)

- 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．ソフトウェアには 2 つの役割が存在し，1 つは導入された端末の状態を常に監視している．端末の CPU 使用率やメモリ使用率，ディスクアクセス等の値を読み取っている．2 つ目は後述するセンサーアントの制御である．センサーアントの持つパラメータに従って，センサーアントの次の転送先を決定している．

次に，センサーアントに関して述べる．この手法では，端末同士が常にアリに見立てたパケットを送受信している．このパケットはセンサーアントと呼ばれ，各々が異なるパラメータを持っている．このパラメータには，主にソフトウェアが読み取った各値に対する閾値が含まれる．もし，センサーアントの持つ閾値を超える値をソフトウェアが報告した場合は，アリの状態が「フェロモン滴下中」という状態へと変化する．この状態になると，センサーアントはフェロモンを滴下しながら端末間を動き回ることになる．他のセンサーアントは，フェロモンを検知するとその方向へ優先的に向かうようにソフトウェアによって制御される．

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

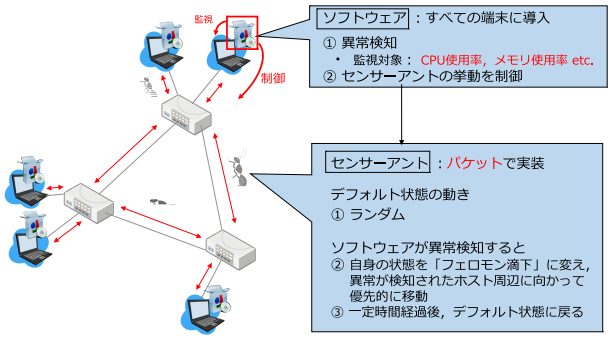

図 3　Ant-Based Cyber Security の概念図

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．

4. 提案手法

本論文では，SDN を用いることで，各端末へ一切手を加えることなく導入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，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，

- 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．

[*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]

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

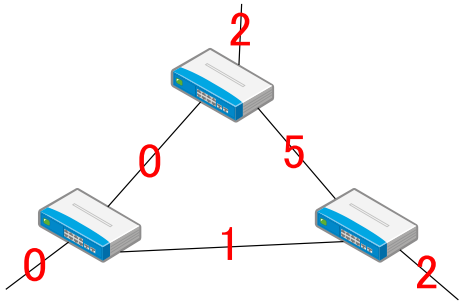

図 4　フェロモンカウント

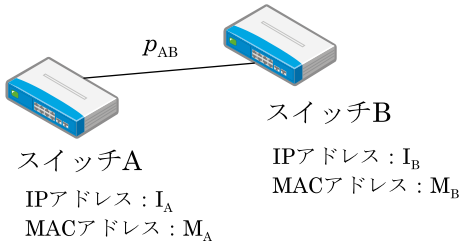

図 5　スイッチ A-B

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．

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

表 4　アントパケットの持つパラメータ

パラメータ名	値	説明
:ant	{0, 1}	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	IPv4 address	パケットのソース IP アドレス

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($[\mathrm{I_B}, \mathrm{M_B}]$) を 1 増やし，スイッチ B も自身の持つ変数 pheromone_count($[\mathrm{I_A}, \mathrm{M_A}]$) 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．

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

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を保証する為には，$e > (\varepsilon)^{\frac{5}{\mathrm{T}}}$ 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．

- :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．

各々のパラメータの扱いは，4.4.3 節にて後述する．

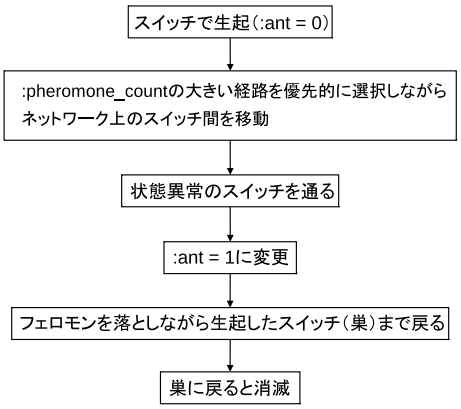

図 6　アントパケットのライフサイクル

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基本的には L3 スイッチとしての機能を有し，それに以下の機能を追加している．

- 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

[*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]

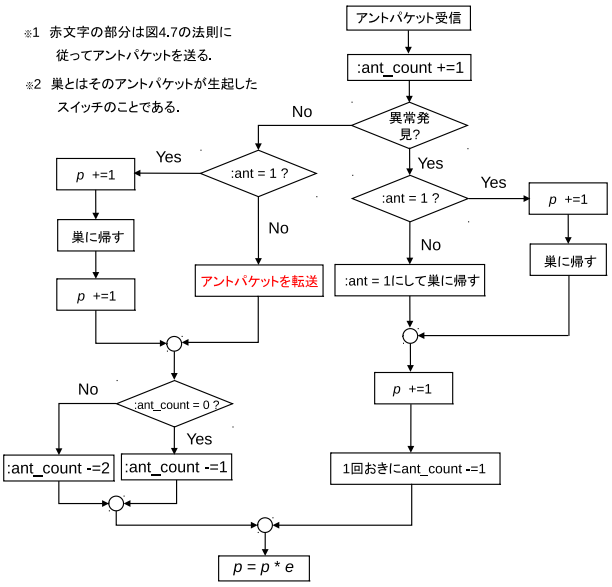

図 7　アントパケットの制御フローチャート

果を指す．図中の p は:pheromone_count の値を表しており，$p = p \times e$ 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．

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．

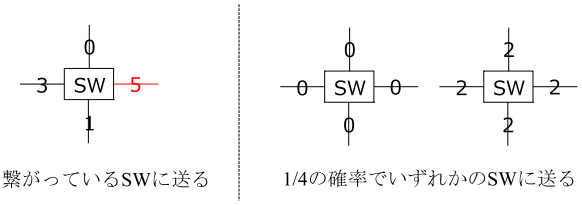

図 8　通常時の「0」アントパケットの制御

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）．

5. 実験

マルウェア検知のために，OpenFlow プロトコルを用いた OpenFlow コントローラを作成し，ポートスキャンを検知できるか実験した．また，CCC DATAset2008[12] を用いて実際に行われた攻撃を検知できるかを実験した．

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: 8GB

5.2 ポートスキャン検知実験

実際にホストに対してポートスキャンを行い検知できるかどうかを実験した．実験はツリー型ネットワークで行い，スイッチ 7 に接続されたホストに対して TCP ハー

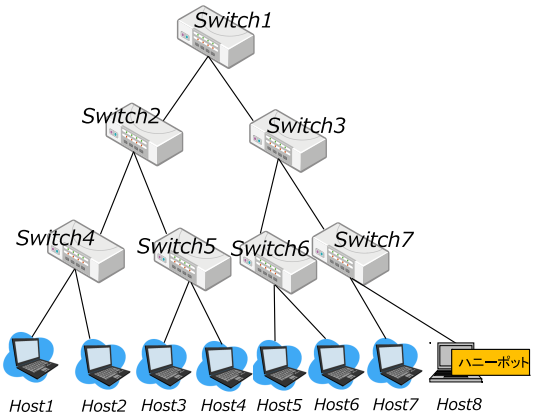

図 9　ボットネット検知実験で用いたネットワーク構成

表 5　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	1	100	or
B	0.98	50	500	or
C	0.98	100	1000	or

フスキャンを 30 秒間隔で行い，実験開始後 10 秒後にスキャンを開始した．この時のアントパケットの生起間隔は 1 秒とし，実験開始後 450 秒でポートスキャンを停止した．フェロモン蒸発係数に関して，4.2 節で示した式において，$T = 600$，$\varepsilon = 0.1$ とし，$e = 0.98$ と設定した．また，異常判定条件は 8[パケット/秒] かつ 500[バイト/秒] とした．

5.3　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．実験に用いた各パラメータ値を表 5 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．

5.4　実験結果と考察

5.4.1　ポートスキャン検知実験

ポートスキャン検知実験の結果を図 10 に示す．ポートスキャン攻撃が行われたホストに接続されたスイッチ 7 の:ant_count が増加しており，検知するまでに時間は 170 秒要した．4.4.3 節で説明した手法で単位時間あたりのパケット数・通信量を監視しているため，最初の 140 秒は異常と見なされない．また，実験開始後 10 秒後にポートスキャンを開始している．従って，合計して最初の 150 秒は:ant_count は増加しないため，アントパケットの集合が遅れたと思われる．

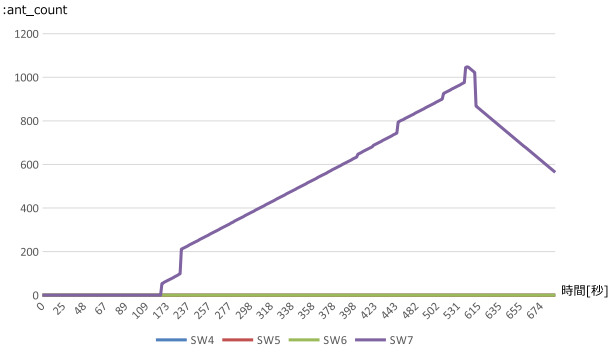

図 10　ポートスキャン検知実験結果

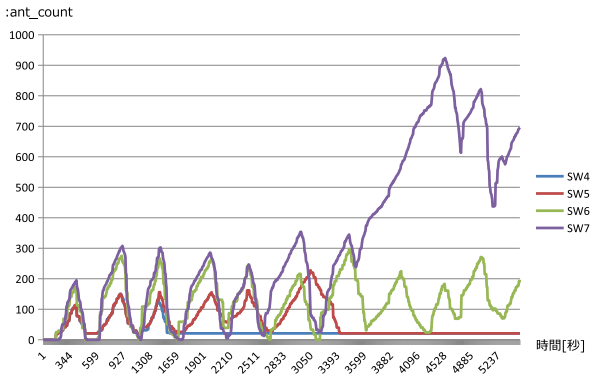

図 11　ボットネット検知実験結果（A）

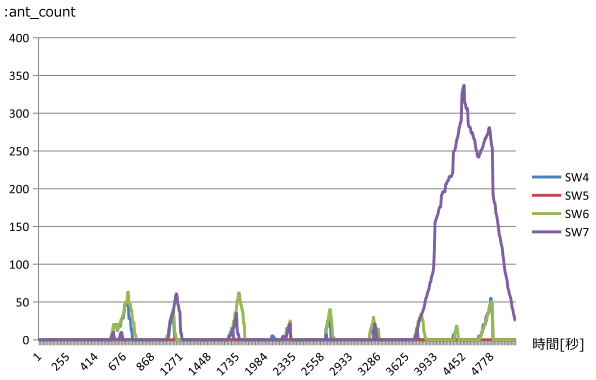

図 12　ボットネット検知実験結果（B）

5.4.2　ボットネット検知実験

ボットネット検知実験の結果を図 11〜図 13 に示す．

- 実験 A について

  1 回目の実験では，3600 秒付近からスイッチ 7 の:ant_count の数が増加し始め，4500 秒付近で最大値を取り，それ以降は増減を繰り返している．2 回目の実験では，3600 秒より前は 1 回目と異なるものの，

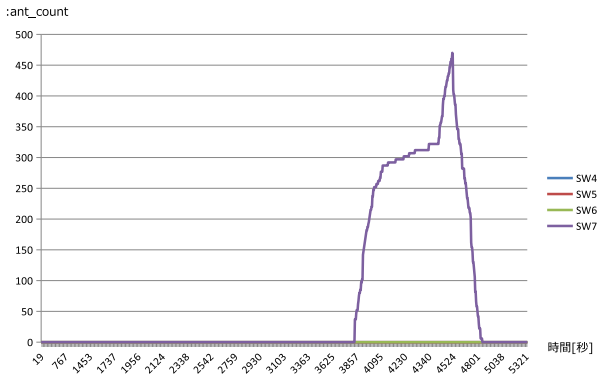

図 13 ボットネット検知実験結果（C）

3600 秒より後は値が増加を続け，4500 秒付近で最大値を取り，それ以降は増減を繰り返すという，1 回目と同じ変化となった．このことから，ボットネットの通信は 3600 秒から 4500 秒の間に最も頻繁に行われたと類推できる．また，1 回目・2 回目共に 4500 秒以降は:ant_count の値が増減を繰り返しながら全体として減少に転じている．加えて，その増減の周期は他のスイッチの増減周期と同期している．これは，:ant_count が本来単調減少していく筈だが，ノイズと見なすべき通信につられてしまい，増加していっていると思われる．即ち，閾値が低いとノイズの影響が強く，ボットネット通信が頻繁に行われたタイミングを正確に知ることが難しい．

- 実験 B について

  閾値を大きくしたことで，ノイズが少なくなっている．スイッチ 7 の:ant_count が増加し始める時間は 3800 秒となっており，実験 A より 200 秒遅くなっている．これは，閾値を高くしたことで，ボットネットの通信が活発な状態へ変化していく際の最初の通信に対して，アントパケットが反応せずに集まらなかったからだと思われる．このことから，閾値を高くすると，ノイズと見なすべき通信に対する精度は上がるものの，検知までの時間がより長くなるというトレードオフが存在していることが分かる．

- 実験 C について

  閾値を実験 B よりも更に大きくしたことで，ノイズと見なすべき通信に対してはアントパケットが集まらず，ハニーポットが接続されたスイッチ 7 にのみアントパケットが集まった．また，:ant_coount が増加し始めるタイミングは，実験 A よりも 300 秒遅い 3900 秒付近であった．

6. 結論

本提案手法を用いて，ボットネット通信データに対して:ant_count が増加することを示した．しかし，本提案手法では:ant_count の増加率は一定であり，図 11 の様に複数のスイッチの:ant_count が同時に増加していくような場合では，いずれのスイッチが攻撃を受けているのか判断することが難しい．また，閾値を大きくすることでノイズに対する精度が向上する一方で，検知に要する時間が長くなるというトレードオフが存在することも判明した．

現在の提案手法では，アントパケットの制御がパケット数と通信量のみに基づくことから，通信を抑えて活動を行う様なマルウェアの検知を行うことは難しい．従って，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，システムをマルチエージェントシステムにすることが今後の課題である．

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] 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.