IPトレースバックを用いた分散協調型DDoS防御手法

水野雅継* 森口一郎**

被害ホストが出力するログを解析してDDoS攻撃を検出し、攻撃ホストを探索して攻撃ホストに最も近いノードで攻撃を抑制するシステムの開発と評価を行った。従来は被害ホスト付近に設置されたファイアーウォールやIPSによって防御を行っていたが、この防御手法では攻撃ホストからファイアーウォールまでの負荷を減少させることはできない。本システムではシグネチャによって攻撃ホストを検出し、トレースバックを行って攻撃ホストからの経路を探索する。探索された経路から攻撃ホスト付近のノード情報を取得し、そのノードで攻撃を抑制することによって被害ホストから攻撃ホストまでのネットワーク負荷を低下させる。しかしログの転送にかかる時間と攻撃情報を他のノードに伝搬する遅延が大きく、反応するまでに時間がかかる。本システムの反応速度が改善され、プロバイダ等のルータに実装されると、DDoS被害の低減が期待できる。

キーワード：DDoS、ファイアーウォール、セキュリティ、トレースバック

Dispersion DDoS Defense Technique by Using IP Trace-back

Masatsugu MIZUNO* and Ichirou MORIGUCHI**

We devised and evaluated a dispersion DDoS defense system. This system detects DDoS attacks by analyzing logs and looks for attacking computers. Usually attacks have been blocked by firewalls or IPSs near servers, however this defense technique cannot reduce malicious traffic between firewalls and servers. Our defense system blocks malicious traffic near assailants. This system detects attacks by signatures and finds the route toward the assailants by trace-back. As a result, this system can reduce that traffic between assailants and servers. However, at present this system takes time to propagate logs and attack information among defense nodes. If these delays will be improved, this system will be expected to reduce DDoS attacks drastically in the Internet world.

Keywords: DDoS, firewall, security, trace-back

*東京情報大学 総合情報学部 情報システム学科学部学生
Tokyo University of Information Sciences, Faculty of Informatics, Department of Information Systems, Undergraduate Student
**東京情報大学 総合情報学部 情報システム学科
Tokyo University of Information Sciences, Faculty of Informatics, Department of Information Systems

2011年11月22日受理

1．はじめに

近年、コンピュータネットワークでは、DoS（Denial of Service）攻撃やDDoS（Distributed DoS）攻撃による被害が深刻になっている。これらの攻撃は、セキュリティホールを利用したものと、リクエストを大量発生させ過負荷を与えるものの２種類に大きく分類することができる。このうちセキュリティホールを利用したものはプログラム作成者の意図しない手法を利用するので、防御は極めて困難である。しかし、セキュリティホールのほとんどは、プログラムの作成元や有志によってセキュリティホールを塞ぐパッチが提供される。このため、セキュリティホールを利用した攻撃は、脅威としてすぐに広まるが、その影響は短期的である。これに対して、過負荷を与える攻撃手法は、正規の利用手法を用い、大量のリクエストを生成して送出する。このため、正規の利用なのか、攻撃であるかどうかの判定が難しく、また送信元を偽装して送出されることもあるため、より判定が難しい。これはサービスを提供し続ける限り、脅威で在り続ける。

これらの攻撃は通常、サービスの提供を行うホストとそれを公開するための回線との間にIDS（Intrusion Detection System）を設置し、管理者がIDSからの攻撃通知を受け手動でファイアーウォール（FW）にルールを追加して防御したり、IPS（Intrusion Prevention System）を設置して自動的に防御させたりする等の対策がとられる。しかし、これらの対策ではDDoSのような意図的に高負荷状態を作り出す攻撃に対し、FWやIPSで防御を行ってもサービスを提供するホストの負荷は減少するが、攻撃ホストからFWやIPSまでの負荷は減少しない。この状態が長く続くとFWやルータ等、ネットワーク機器の処理能力飽和を誘発し、輻輳が発生する。また、帯域の圧迫や遅延、パケットロスなどにより応答速度の悪化などサービス品質が低下し、他の利用者に対しても影響を与える。このような影響を最小限にするには、攻撃ホストに近い位置で攻撃を抑制することが効果的である。攻撃ホストに近い位置で攻撃を抑制すると、サービスを提供するホストだけではなく、攻撃ホストからサービスを提供するホストまでの回線やネットワーク機器の負荷を抑えることができる。

本研究では、故意に過負荷状態を作り出すDoS・DDoS攻撃に対し、攻撃ホストを特定して攻撃ホストに最も近いノード（ルータ等、ネットワークの節点を指す）で攻撃を止めることにより、ネットワークそのものへの被害低減と、被害ホストが正常に稼働し続けることを目的としたシステムの提案と構築・評価を行った。

なお、このシステムはNTTの開発したMoving Firewall[1]と類似しているが、本研究で使用している攻撃判定手法や攻撃ホスト検出手法、使用するプロトコルなどは本研究を進めるにあたって新たに開発・実装したものである。Moving Firewallでは専用装置とオペレータによる操作が必要である。しかし、本システムは汎用のLinuxにソフトウェアを導入するだけで使用可能であり、オペレータの介入を必要とせず、また特別な設定をしなくても自動的に防御する点で大きく異なる。

2．システムの概略

多くのアプリケーションは、クライアントが接続してきたり、何らかの処理を行ったりすると、ログを出力する。本システムはそのログをリアルタイムで解析し、ファイアーウォールに対してルールの追加と削除を行う。これらはLinuxをルータとして動作させ、その上にJavaで作成した制御ソフトウェアとファイアーウォールやログの転送を行うソフトウェアを組み合わせて実装した。

本システムでは下記の手順で攻撃を抑制する。

攻撃ホストから被害ホストに対して攻撃が行

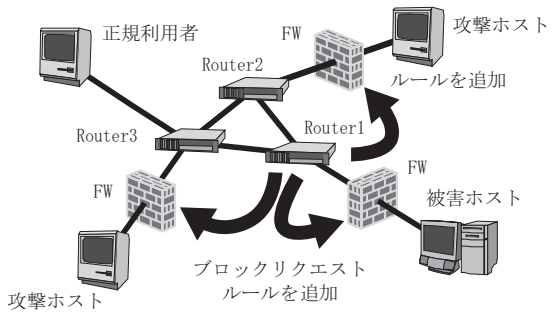

図１　システム構成概略図

われると、被害ホストのログが本システムを実装したRouter1 に対して送信される（図１）。Router1 はログの内容やその発生頻度から攻撃であるかどうかを判定する。攻撃であると判定した場合は、被害ホストとRouter1 の間にあるFWに対して該当の通信をブロックするためのルールを追加する。その後、攻撃ホストから被害ホストまでの経路を探索する。攻撃ホストから被害ホストまではRouter2 とRouter3 を経由するため、これらに対してRouter1 からブロック要求が送信される。これらのルータはそれぞれが接続しているFWに対して攻撃ホストからの通信を止めるルールを追加する。

３．トレースバック手法

トレースバックとは、パケットがどのような経路を通過してきたのかを順に辿る行為を意味する。本システムでは攻撃ホストの位置を検出するために、攻撃ホストが送信してきたパケットに対してトレースバックを行う。

パケットの送信元IPアドレスが偽装されておらず、往路と復路が対称であれば、基本的にOS付属のコマンドで送信元からの経路を調べることができる。しかし、送信元IPアドレスが偽装されたパケットの場合、そのIPアドレスを用いて経路探索を行うと、コマンド自体は正常に終了するかもしれないが、偽りの経路が出力される可能性が極めて高い。また、パケットの送信元IPアドレスが偽装されているのか、されていないのか判断することができないため、送信元IPアドレスを信頼することができない。DoS攻撃やDDoS攻撃は送信元アドレスを偽装して攻撃してくることが多いので、OS標準のコマンドでトレースバックを行うことは困難である。

この解決策として、本研究ではMACアドレスを用いてトレースバックを行った。この手法は、播磨宏和らによるトレースバック手法を参考にした[2]。

現在、ほとんどのネットワークはLayer2 にEthernet、Layer3 にIPを利用し、それらを組み合わせて構成されている。IPパケット内の送信元IPアドレスは自己申告制であるので容易に偽ることができる。これに対してEthernetで利用されるMACアドレスは、偽装自体は可能であるが、送信元アドレスも宛先アドレスも同じネットワーク内でしか意味をなさない。このため、偽装できるのは送信者から送信者が接続しているルータまでの間のみであり、一般に偽装はそれほど大きな意味をなさない。またすべてのパケットに対してMACアドレスの偽装を施すとしても、経由するLayer3 ルーティング機器をすべて支配下に置く必要があり、事実上不可能である。本研究では攻撃ホストの所属するネットワーク以外で送信元MACアドレスの偽装が困難であることを利用してトレースバックを行う。

トレースバックを行う事前準備として、各ルータにパケットが通過する時、送信元MACアドレスと送信元IPアドレスをハッシュ表に記録する。トレースバックを行う際には送信元IPアドレスをキーにMACアドレスを探索し、該当IPアドレスのトレースをそのMACアドレスを持つルータに対して依頼する。これを再帰的に行い、該当MACアドレスを持つホストが存在しないか、トレースのリクエストを解釈できないルータに遭遇するまで行う。この手法により、該当IPアドレスを持つホスト付近までの経路を探索することが可能となる。

4．ログの転送手法

本システムでは、ログを解析して攻撃の判定を行う。ログの受け渡しには、通常Syslogと呼ばれるプロトコルが利用される。本システムではCentOS 5.6を利用したが、このOSはログの管理と転送に標準でsyslogdを用いる。syslogdはUDPを利用してSyslogメッセージを転送するため、ホストやネットワークに対して負荷がかかるとログが失われることがある。ログが失われると本システムで攻撃を検出することが出来なくなるため、この問題は致命的である。また、別の問題としてsyslogdではログを十分に分離できないという問題がある。Syslogはファシリティとプライオリティという２つのパラメータを持つ。ファシリティはログの種類、プライオリティはログの重要度を表し、syslogdはこれら２つのパラメータによってログを分離する。しかし、ログを送信するアプリケーションの組み合わせによってはファシリティとプライオリティが衝突し、複数のログが混ざり合うことある。ログが混ざり合った状態でも、本システムはログに適切なプレフィクスさえあえばシステム内で分離させることが可能である。しかし、これでは不要なログを転送することになるので、余計な伝送帯域や処理時間を必要とする。

これらの問題を解決するためにsyslog-ngを利用する。syslog-ngはsyslogdより高度な管理機能を持つログメッセージ管理ソフトウェアである。syslog-ngはTCPによるログ転送や、パターンマッチによるログの分類などの機能を持つ。

ログの転送はsyslog-ngのTCPによるログ転送機能を用いる。これによって、ログが紛失する可能性はUDPを使用している場合と比較して格段に低くなる。

また、パターンマッチによってログを分離できるので、ログを出力するソフトウェアで適切なプレフィクスさえ付加すれば、プライオリティとファシリティが衝突した場合でもログを分類することができる。これによって余計なトラフィックを発生させることなくログを転送することができる。

5．NIO（New IO）による非同期通信

本研究では、主にJavaを用いてシステムが構築されている。

Javaで通信を行うプログラムを作成する時、一般にはjava.net.Socketを用いる。このSocketは通信相手ごとに１つのインスタンスが必要である。複数の通信相手と同時に通信を行う場合は、Threadクラスを利用して通信相手の数だけスレッドを作成し、その上でSocketを扱う必要がある。

スレッドはプログラムの並列化を行う上で不可欠な要素であるが、生成には少なからずCPUの処理時間やシステムのリソースを消費する。また、スレッドの適度な利用はコンピュータの利用効率向上に繋がるが、多すぎるスレッドはオーバーヘッドが増加して効率が低下することがある。

その解決策として、JavaではNIO（New IO）と呼ばれるパッケージを提供している。このパッケージは入出力に関わるパッケージで、標準の入出力にはないノンブロッキングモードをサポートする。

ノンブロッキングモードとは、通信相手との同期を取らずに処理を行うモードである。例えばソケットを利用してクライアントの接続を待ち受けるとき、通常はクライアントから接続されるまでプログラムは停止する。ノンブロッキングモードを利用すると、プログラムは停止せず、次の処理へ進む。

本システムでは積極的にNIOを利用し、１つのスレッドで複数のクライアントと通信を可能にしている。

6．DDoS防御システム

6.1　システム概要

今回作成したシステムは、以下のモジュール

から成る。

・通信モジュール
・ログ解析モジュール
・攻撃判定モジュール
・FW操作モジュール
・トレースバックモジュール

このシステムはLinux上で構築されており、Syslogを解析することによって、攻撃の判定やブロックを行う。本システムの基本的な処理の流れを図２に示す。

被害ホストがパケットを受信したり、被害ホスト内のサーバソフトウェアが処理をしたりすると、Syslogに対してログを書き込む。書き込まれたSyslogはTCPを用いて本システムに転送され、Syslog受信モジュールにて受信する。受信したSyslogをログ解析モジュールに渡し、プログラム内部で使用するためのフォーマットに変換する。変換後、Protocol Queueにてキューイングし、攻撃判定モジュールに渡す。攻撃判

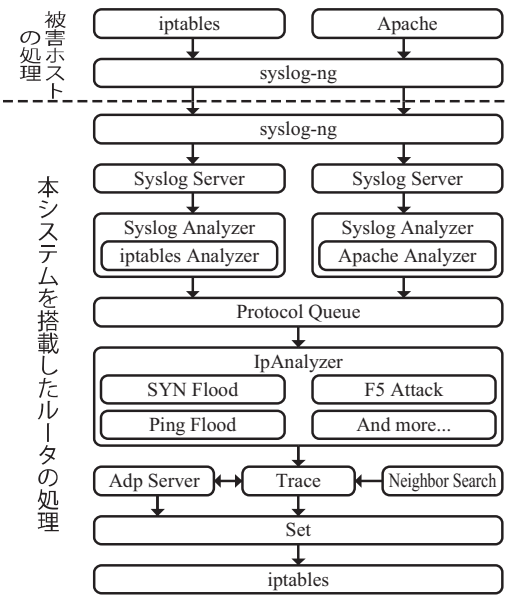

図２　システムの基本フロー

シグネチャにマッチした場合、トレースバックを行うTraceとFWの設定を行うSetに処理が流れる。

定モジュールは、登録されたシグネチャを利用して攻撃判定を行う。攻撃であると判定すれば、FW操作モジュールによってFWに対しルールを挿入する。

その後、隣接探索モジュールで得た情報を元にトレースバックモジュールにて攻撃元を検出し、攻撃元付近のルータに対し、ノード間通信モジュールを経由してルールの挿入を依頼する。この作業をリアルタイムで行う。

6. 2　通信モジュール

通信モジュールは、Syslog受信モジュール、ノード間通信モジュール、隣接探索モジュールの３つに分けられる。Syslog受信モジュールとノード間通信モジュールはNIOを、マルチキャスト送受信モジュールはMulticastSocketを利用して構成されている。

6. 2. 1　Syslog受信モジュール（Syslog Server）

本システムでは、ログメッセージをTCPによって受信する。通常、Linuxはsyslogdを介しUDPでログの転送を行うが、本システムはログを損失すると正しく攻撃が判定できないためsyslog-ngを利用し、TCPでログの受信を行う。従って、iptablesやApacheなどのアプリケーションから出力されるログは、一旦syslog-ngに渡され、プレフィクスによる分類を行った後に本システムへ転送される。

Syslog受信モジュールでは、syslog-ngからのログを受け取り、ログ解析モジュールへと渡す。このモジュールはNIOを利用して複数のクライアントから同時にログを受信できるよう設計した。NIOを利用することで１つのスレッドのみで複数のクライアントからログを受け取ることができる。Syslog受信モジュールはこのNIOを利用し、受信バッファに貯めたデータ内で終端文字を発見次第ログ解析モジュールに渡す。

6. 2. 2　ノード間通信モジュール（Adp Server）

ノード間の通信においてもNIOを使用す

る。ノード間で通信を行うには、何らかのプロトコルを用意する必要があるが、本研究ではWebで利用されるHTTP（Hypertext Transfer Protocol）と類似したテキストベースの独自プロトコルを使用している。このプロトコルは、Layer4 にTCPとUDPを使用することができ、信頼性はTCPとUDPどちらのプロトコルを利用するかに依存する。高い信頼性は必要なく、低い負荷と応答性を重視する場合はUDPを、多少負荷が高くても信頼性を重視したい場合はTCPを用いる。本研究では隣接関係の探索にUDPを利用し、それ以外ではTCPを利用している。

ノード同士はこのプロトコルを用いてメッセージを相互に送受信する。本プロトコルはリクエスト－レスポンス型のプロトコルであり、いずれかがリクエストを送信すると、相手がレスポンスを返すかタイムアウトが発生するまで、どちらもリクエストを送信することはできない制約を持つ。

HTTPと同様、メッセージはリクエスト行あるいはレスポンス行・ヘッダ・メッセージボディの３つから構成される。リクエストメッセージの例を図３、レスポンスメッセージの例を図４に記す。

リクエストメッセージには、どのようなリクエストなのかを識別するためのメソッドが定義されている。本システムでは、攻撃元までの経路を探索するTRACEメソッドや、ノード情報を取得するSCANメソッド、ファイアーウォールルールの追加を要求するSETメソッドを独自に実装している。

レスポンスメッセージは、ステータス番号といくつかのヘッダ、メッセージボディから成る。ステータス番号は概ねHTTPと同様の意味を持つ。例えば、リクエストが正常に終了すれば200を、リクエスト構文に問題があれば400を返す。

メッセージボディの内容はContent-Typeヘッダによって決定される。Content-Typeには、ルート情報の伝送に用いるRouteと、ファイアーウォールルールの伝送に用いるFirewallRuleの二つを定義した。

RouteやFirewallRuleで利用されるフォーマットはXMLに類似したフォーマットである。このフォーマットは「＜」から「＞」までを１つのタグとして解釈する。以下はこのフォーマットの標準的な書式である。

```
＜要素　属性＝"値"＞内容＜/要素＞
```

タグには要素の存在が必須である。要素はそのタグの意味を決定する。

属性は要素名に対して付加情報を設定する時に使用する。これは要素と異なり必須ではないので省略可能である。書式例では１つの属性しか記述していないが、スペースを区切り文字として複数の属性を記述することもできる。値は必ずダブルクオーテーション（"）で囲う必要がある。

内容は要素を適用する対象である。内容の中にタグを記述し、入れ子構造にすることも可能

```
① TRACE 192.168.250.1 ADP/1.0
② Host:Router01
  Group:GroupName
```

図３　リクエストメッセージの例

①はリクエスト行、②はヘッダを表す。

```
① ADP/1.0 200 OK
② Host:Router01
  Content-Type:Route

  <Route HopCount="3" Destination="192.168.250.1">
  <Via Count="1" Address="192.168.250.14" />
③ <Via Count="2" Address="192.168.250.18" />
  <Via Count="3" Address="192.168.250.22" />
  </Route>
```

図４　レスポンスメッセージの例

①はレスポンス行、②はヘッダ、③はメッセージボディを表す。

である。

タグは種類あり、「＜要素　属性＝"値"＞」を開始タグ、「＜/要素＞」を終了タグと呼ぶ。要素によっては、内容を持つ必要がないものも存在する。そのような要素に対して終了タグを記述するのは冗長である。そのため、終了タグを省略できる省略タグとして「＜要素　属性＝"値"/＞」を用意した。図４の例では、ルート情報を伝達する際にこのプロトコルを利用している。ルート情報を転送するときは、Content-TypeにRouteを指定し、メッセージボディに要素「Route」を指定する。その属性として総ホップ数を表す属性「HopCount」と、宛先を表す属性「Destination」を指定している。実際の経路は要素「Via」によって記述されている。「Via」は内容を持たないので省略タグとして記述し、属性「Count」にホップ数、属性「Address」に経由ノードのアドレスを記述する。

6. 2. 3　隣接探索モジュール（Neighbor Search）

隣接探索モジュールはマルチキャストを用いて隣接ノードの情報を送受信し、隣接関係を探索する。情報の送受信には、６．２．２で説明したプロトコルを使用する。マルチキャストを使う都合上、Layer4 はUDPとなる。デフォルトでは起動時に１回、その後は60秒に１回マルチキャストでSCANメソッドを利用し、隣接ノードに自身のノード情報をリクエストする。このリクエストはUDPという特性上、必ずしも受信される訳ではない。このため、パケットが破棄されにくいネットワーク環境ではこの間隔を長く、破棄されやすい環境では短くすると効率がよい。

SCANメソッドを利用すると、自身の情報が自動的に付加され、これをマルチキャスト送信する。リクエストを受け取ったノードは、受け取ったパケットからノード情報を取得し、レスポンスとして自身の情報を返信する。この作業を行うことにより、隣接ノード同士が互いの情報を交換し、存在を確認し合うことができる。

通常、マルチキャストはノード同士が互いに送受信しあうことはなく、複数の物理インターフェースから同時に流すこともない。しかし、ルータのように複数の物理インターフェースが存在するシステムでは、それぞれのインターフェースからマルチキャストパケットを送信する必要がある。本システムではJavaのjava.net.MulticastSocketを利用してマルチキャストを扱う。このMulticastSocketを利用して複数のインターフェースからマルチキャストパケットを送信するには、インターフェースの数だけMulticastSocketのインスタンスを用意する必要がある。

マルチキャストを送信する条件として、マルチキャストパケット用のルートを設定しておく必要がある。ルートが設定されていない場合、java.net.SocketExceptionが発生する。デフォルトゲートウェイが設定されている場合、マルチキャストパケットはデフォルトゲートウェイ宛に流れる。通常ならば単一ルートの設定だけでよいが、複数のインターフェースを持つマシンには対応できない。そこで、マルチキャストを送信するすべてのインターフェースにマルチキャストアドレス宛のルートをルーティングテーブルに登録する。Linuxにはルートを設定するための手段がいくつも用意されているが、例えば、

```
# route add -net 224.0.0.0/4 dev［インターフェース名］
```

のようなコマンドを使用して、同じ経路を複数のインターフェースに適用することができる。

このようにすることで、すべてのインターフェースからマルチキャストパケットを送信できるようになる。

上記の手順でマルチキャストの送信は可能であるが、マルチキャストを受信するためにはさらにいくらかの手順を踏む必要がある。

本研究では、トレースバックを行う際に隣接ノードと隣接ノードが接続しているインター

フェースを対応付ける必要があった。通常であれば、マルチキャストはMulticastSocketで送受信を行うが、MulticastSocketでマルチキャストパケットを受け取ると、どのインターフェースでパケットを受信したのか判別することができない。

このため、本システムではマルチキャストの受信にMulticastSocketではなくUDPを扱うDatagramSocketを利用する。DatagramSocketは本来UDPユニキャスト通信を行うために用いるが、Layer4がUDPであれば、マルチキャストでも受信することができる。このDatagramSocketをインターフェースごとに生成して待機させることにより、マルチキャストパケットを受信し、それをどのインターフェースから受信したのか識別することができる。この使い方は本来の使用方法と異なるのでMulticastSocketが自動的に行なってくれる設定を手動で行う必要がある。

マルチキャストはLayer2 とLayer3 でそれぞれ独自のアドレスを持つ。通常ネットワークインターフェースは、受信したパケットの宛先MACアドレスが自分の持つMACアドレスと一致するかどうかを確認する。一致すればパケットを取り込み、不一致ならばパケットを破棄する。マルチキャスト利用時は、ネットワークインターフェースが持つMACアドレスとLayer2 マルチキャストアドレスの両方とマッチを行う。このLayer2 マルチキャストアドレスは、MulticastSocketを利用すると自動的に設定される。しかし、DatagramSocketの場合はこれを手動で設定する必要がある。もし、設定を行わない状態でマルチキャストパケットを受け取ると、パケットの宛先MACアドレスがマルチキャストアドレスであるため、受信インターフェースのMACアドレスとマッチせず、パケットが破棄されてしまう。よって正しく受信するためには、インターフェースをプロミスキャスモードで待機させるか、手動でインターフェースにマルチキャストアドレスを割り当てておく必要がある。例えば、マルチキャストアドレスをインターフェースに割り当てるには以下の書式を利用する。

```
# ip maddr add［Layer2 マルチキャストアドレス］dev［インターフェース名］
```

上記の書式を利用すると、インターフェースにマルチキャストアドレスが登録され、マルチキャストパケットがインターフェースで破棄されることはなくなる。しかし、インターフェースのLayer3 アドレスとマルチキャストパケットの宛先アドレスが異なるためやはり破棄されてしまう。MulticastSocket利用時であれば、適切なインターフェースを１つ選択し、自動的にLayer3 マルチキャストアドレスが付加される。しかし、自動設定では同じマルチキャストアドレスを複数のインターフェースに付加することはできず、どのインターフェースでパケットを受け取ったのか識別することができない。また、手動ですべてのインターフェースに対しLayer3 マルチキャストアドレスを割り当てても、受信パケットはいずれか１つのインターフェースからしか入らない。

そこで、宛先アドレスをLayer3 マルチキャストアドレスからインターフェースのLayer3 アドレスへ書き換え、マルチキャストアドレスをユニキャストアドレスに変換する。この書き換えにはiptablesを利用する。iptablesはデフォルトですべてのパケットを一度natテーブルのPREROUTINGチェインに通す。従ってPREROUTINGチェインを通過する際に、宛先を書き換えるDNATターゲットを利用して宛先Layer 3 アドレスを書き換える。具体的には以下の書式を利用する。

```
# iptables -t nat -A PREROUTING -i［インターフェース名］-d［マルチキャストアドレス］-j DNAT -to［IF Layer3 アドレス］
```

上記の書式に従いパケットが入ってくるインターフェース名・変換するマルチキャストアドレス・インターフェースのLayer3 アドレスを

それぞれ置き換え、マルチキャストを受信するインターフェースすべてに対して適用する。これらの手順によってマルチキャストを受け取ることができる。

6. 3 ログ解析モジュール（Syslog Analyzer）

ログ解析モジュールはSyslogメッセージを解析する。現時点では、iptables及びApacheのログを解析することができる。これらのログは、正規表現によるパターンマッチングによって各要素を切り出し、システム内で取り扱いやすいようLayerごとに分けて格納される。ここでは、例としてiptablesのログを処理する方法について説明する。

iptablesのログは、要素がスペースで区切られているので、一旦スペースで分割する。分割された値は、次の正規表現とマッチを試みる。この正規表現には以下の書式を用いる。

```
([^=]+)=([^=]*)
```

この正規表現にマッチすると、「=」で区切られた両辺を取得することができる。左辺が要素名を表し、右辺が値であるので、左辺の要素名を元に解析を進める。Javaの正規表現エンジンは、一度マッチを行うと、次回はその次の文字からマッチを再開するので、同じ正規表現をループ文の中に入れるだけですべての要素を取り出すことができる。

図５の例では、最初に要素名IN、値eth1 が得られる。次の要素は要素名OUTだが、「=」の先が存在しないため、値はnullが得られる。このようにして右辺と左辺が分けられ、構文解析が行われる。

構文解析は、抽出した要素名がどのレイヤーに属するのかをハッシュ表を用いて判定する。

```
IN=eth1 OUT= MAC=00:0a:79:31:03:ac:00:07:e9:54:67:82:
08:00 SRC=192.168.200.254 DST=192.168.200.1 LEN=60
 TOS=0x00 PREC=0x00 TTL=128 ID=57552 PROTO=ICM
P TYPE=8 CODE=0 ID=1024 SEQ=6656
```

図５ iptablesのログの例

ハッシュ表に要素名が登録されていれば、その要素名が所属するレイヤーが返却されるので、そのレイヤーに対して値を設定する。これをすべての要素に対して行う。

6. 4 攻撃判定モジュール（Ip Analyzer）

攻撃の判定は、SYN FloodやPing Floodの場合、本システムが単位時間あたりに受けたログの数を計数することによって行う。

ログ解析モジュールによって得たパケット情報の宛先が保護対象のホストであった場合、逐次シグネチャに渡し、攻撃判定を行う。

一度パケット情報を得ると、その情報がおのおののシグネチャ内で一時保管される。一定時間経過すると、それらは自動的に削除されるが、時間が経過する前に同じパケット情報を得ると、カウンタを増加させる。このカウンタが一定数を超えた時点で攻撃と判定する。攻撃であると判定した場合、違反カウンタを１増加させる。この違反カウンタの値によって対象をブロックする時間が決定される。ブロックする時間はシグネチャごとに設定可能だが、Ping Floodを検出するシグネチャの場合、初期値は初回検出時に30秒、以後カウンタの値が増加するに伴い、60、120、360、43200秒となるように設定されている。初期値は30秒と比較的短く設定してあるのは、正規利用者を誤検出した場合にブロックしてしまう時間を短くするためである。しかし、保護するシステムの負荷やユーザ数によって最適値が異なるので、これらの値は環境に合わせて変更する必要がある。

6. 5 FW操作モジュール（Set）

本システムは主にJavaを利用して構成されているが、Javaだけではコンピュータを流れるパケットのフィルタリングはできない。そのため、Linuxのパットフィルタ実装であるiptablesを利用してパケットをフィルタリングする。この構造は異なるシステム上でプログラムを動作させる際、このモジュールとコマンドの記述方法の変更のみで適用可能になるというメリットを持つ。

FW制御モジュールは、iptablesのmangleテーブル内にRULEという名のユーザ定義チェインが存在するという前提で動作する。これはRULEチェインの中のみにスコープを限定し、ユーザが手動で作成・挿入したルールに対して影響を与えないようにするためである。mangleテーブル内に存在する必要があるのは、mangleテーブルがどのテーブルよりも先にパケットとマッチされるからである。本システムが自動的に設定するルールよりユーザが作成したルールを優先させたい場合は、より後ろのテーブルにRULEチェインを移動するか、RULEチェインの前にルールを追加すればよい。なお、このRULEチェインは制御プログラム起動時と終了時に初期化される。

Javaからiptablesを利用するには、ProcessBuilderを利用する。ProcessBuilderは外部プログラムを実行するためのクラスである。このクラスを経由して、FWの初期化や挿入、削除の操作を行う。

ルール挿入のリクエストがあれば、ルールの破棄予定時間を確認し、破棄予定時間が負数でなければルールを挿入する。挿入後は定期的に破棄時間を確認し、破棄時間を経過したルールは削除する。同じルールが存在した場合、ルール自体は操作せず、破棄時間を上書きする。

6.6　トレースバックモジュール（Trace）

トレースバックモジュールは、3章で説明した手法を実装した。隣接探索モジュールで得た隣接情報を元に、取得したパケットの送信元MACアドレスと一致する隣接ノードを検索する。発見された隣接ノードに対してTRACEメソッドを利用し、トレースバックのリクエストを行う。この作業を再帰的に行うことによって、攻撃ホストまでの経路を特定する。

経路が特定されると、ノード間通信モジュールによって、攻撃ホストに近いノードから被害ホストに向かって順にブロックリクエストを行う。正しくリクエストが受理された地点でリクエストを打ち切る。

7．実験と考察

7.1　実験環境

7.1.1　コンピュータ環境

すべてのコンピュータはCentOS 5.6 Kernel i686 2.6.18-238.12.1をオペレーティングシステムとしている。ルータとなるノードには、Javaが動作するようOracle Java Runtime Environment 1.6.0 Update 26をインストールし、プログラムの動作基盤とした。以下に使用したマシンのスペックを記す。

Router01
　CPU: Intel Celeron D 3.06GHz
　Memory: 512MB
　NIC: Realtek RTL8111B Gigabit Ethernet（onboard）

Router02（IBM ThinkPad R32）
　CPU: Pentium4 Mobile 1.60GHz
　Memory: 512MB
　NIC: Intel PRO/100 VE Fast Ethernet（onboard）

Router03（Oracle VirtualBox上）
　CPU: Intel Core i5 3.33GHz
　Memory: 192MB
　NIC: Intel 82578DC Gigabit Ethernet（Intel 82543GC Gigabit Ethernet Emulation）

Server
　CPU: Intel Core2 Duo 2.20GHz
　Memory: 2GB
　NIC: Intel 82573L Gigabit Ethernet（onboard）

Client（IBM ThinkPad i1200）
　CPU Intel Celeron 700MHz
　Memory: 192MB
　NIC: Realtek RTL8169 Gigabit Ethernet（Cardbus）

7. 1. 2 ネットワーク環境

Layer 2 Switch

Cisco Catalyst 2970G-24T（24port Layer2 Gigabit Ethernet Switch）

コンピュータをルータとして動作させるためには、少なくともNICが2枚必要だが、用意できなかったためTagged VLAN（IEEE 802.1Q）を利用した。これにより、1つのNICで複数のネットワークを扱えるようになる。Tagged VLANを使用するためにはスイッチがこれに対応している必要がある。このため、Tagged VLANに対応しているCatalyst 2970G-24TにVLANを設定し、すべてのコンピュータをこれに接続した。これによって物理的にはスター型になるが、論理的にはVLANを利用したPoint-to-Point接続になる。

7. 2 プログラム仕様

7. 2. 1 使用言語

Java 1.6.0 Update26

CentOS 5.6にはデフォルトでOpenJDKがインストールされている。しかしNIOの挙動が安定しない場合があることからOpenJDKをアンインストールした後にOracleのJDKをインストールした。

7. 2. 2 関連プログラム

iptables 1.4.7

パケットの通過したログ取得や、パケットフィルタリングを行う。

CentOS 5.6は、標準でiptables 1.3.5がインストールされている。しかし、iptablesの挙動が安定しないことがあったため、配布元から1.4.7をダウンロードし、1.3.5のspecファイルを改変してrpmパッケージを作成、インストールを行った。

syslog-ng 3.2.2

ログのフィルタリングと転送を行う。

syslog-ngはCentOS 5.6に標準で含まれていないため、開発元のBalaBitよりパッケージを取得し、インストールを行った。

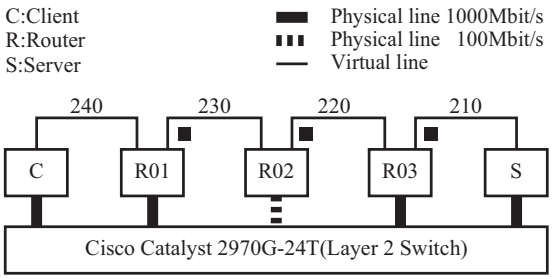

図6 ネットワーク構成

太い実線と鎖線は物理回線、細い実線はVLANによって作られた仮想回線を意味する。仮想回線の上にある数字はVLANを識別するVLAN IDである。

7. 3 時間計測

時間の計測は、各々のマシンでtcpdumpを実行し、そのログを採取した。ログの採取は、図6の■で示すインターフェースにて行った。

攻撃を行うとそのパケットがログとして記録されるが、システムによってブロックされるとそのログが途切れる。これを利用し、ログが途切れた時間からログが記録され始めた時間を引き、その時間を反応時間として計測した。

7. 4 検証手法

攻撃はhpingを用いてClientからServerに対して行った。hpingは、様々なパケットを生成して送信することができる負荷テストプログラムである。このプログラムを利用してPing Flood及びSYN Floodの両方で、1秒当たり500回の頻度で攻撃を行った。この攻撃頻度はhpingと使用したマシンの制約によるものである。Ping FloodではICMP Echo Requestを、SYN FloodではTCPのSYNフラグを立てたパケットを生成して送信し、攻撃は送信元を適当なアドレスに偽装した状態で行った。

攻撃には以下のコマンドを用いた。コマンド内の10.10.10.10は偽装したホストアドレスである。

```
# hping -1 -i u1000 -a 10.10.10.10 server
# hping -i u1000 -S -a 10.10.10.10 -s 9191 -k server
```

7.5　実験結果

被害ホストに最も近いRouter03では、SYN Floodで110ms、Ping Floodで172ms程度、停止するまでに時間がかかっている（表１）。この時間のほとんどはsyslog-ngからシステムへログを転送するまでの時間であると思われる。

攻撃ホストに近いRouter01では、SYN Floodで460ms、Ping Floodで519ms程度、停止させるまでに必要としている。これは、TCPの接続確立処理大きく関係していると思われる。

ノードに対してリクエストを送るには、TCPで接続を確立しておく必要がある。このため、トレースバック時には隣接ノードに対し、接続を確立してからトレースリクエストを送信している。これは、ブロックリクエストを送信する場合においても同様で、リクエストを送信する直前にコネクションを確立する。このTCPコネクション確立処理とトレースバックによるオーバーヘッドの影響で、反応にかかる時間が延びていると推測できる。

別の原因として、攻撃によりネットワークが輻輳を起こしていることが挙げられる。この確認のために、攻撃頻度を 1/10程度に下げると、反応時間も 1/10程度になり、攻撃頻度を増加させると反応時間は長くなった。このような輻輳状態ではログの転送やノード間通信に遅延が発生する。特に、トレースバック時の遅延は致命的で、輻輳の影響を大きく受ける。

この解決策として、QoSを利用し、本システムの通信を優先してネットワークに流す手法が考えられる。

表１　実験結果

反応時間とは、ログが途切れた時間からログが記録され始めた時間の差を意味する。サンプル数は５である。

	Router01		Router03	
	反応時間	標準偏差	反応時間	標準偏差
SYN Flood	460	12777.3	110.4	218.8
Ping Flood	519.7	206.7	172.5	163.6

（ms）

また、現在のシステムでは、トレースバックとルールの設定を別々に行っているが、トレースバックを行いながらルールを設定するようにすれば、遅延の短縮が期待できる。

本システムの反応にかかる遅延は、大量のパケットを送出するFlood系のDoS攻撃を抑制する上でほとんど無視でき、実用上十分な性能を持つ。しかし、DDoS攻撃においては攻撃を抑制するまでのわずかな間でも複数のホストより攻撃を受けるので、十分に攻撃を抑制できない可能性がある。また攻撃の中には数パケットでホストを停止させてしまう攻撃もあるため、攻撃を広く抑制するには反応速度の改善が必要である。

8．まとめ

本研究では、DDoS攻撃に対し、攻撃ホストに近いルータで攻撃を抑制するシステムの提案と評価を行った。

本システムはLinux上で動作するため、特別な機器は必要なく、Linuxが動作するコンピュータさえあれば導入することができる。

本システムはログを解析して動作し、攻撃であると判定した場合にトレースバックを行う。トレースバック結果を元に攻撃ホスト付近でのブロックを行い攻撃の抑制を行う。攻撃ホストまでのトレースバックは送信元MACアドレスを用い、送信元IPアドレスが偽装されていても正しく攻撃ホストまでの経路を辿ることができる。

実際に攻撃を行い、このシステムの反応時間を計測した結果、攻撃ホスト付近でも比較的短時間に攻撃を抑制できたが輻輳による反応遅延も確認された。

このシステムを実用化にするためには、まず輻輳による遅延の影響を最小化し、攻撃を検出するシグネチャをさらに充実させる必要がある。これらを改善した後に各プロバイダのルータやデータセンタ等のエッジルータに実装されると、DDoS攻撃によるホストの被害低減と、ネットワーク全体の負荷減少が期待できる。

【参考文献（References）】

［1］日本電信電話株式会社：攻撃元にまで攻め上がりながらネットワーク全体を防御するDDoS攻撃対策システム「Moving Firewall」を開発
http://www.ntt.co.jp/news/news03/0302/030218.html

［2］播磨宏和：情報処理学会第67回全国大会講演論文集，Mar.2005
MACアドレスを用いたIPトレースバック技術の提案
http://www.wata-lab.meijo-u.ac.jp/file/convention/2004/200503-IPSJ_NC-Hirokazu_Harima.pdf