攻撃側ネットワークの制御による
DoS攻撃への加担を防ぐ研究

情13-243 高岡 奈央

目 次

図 目 次

表 目 次

はじめに

サイバー攻撃によるサービス障害は年々増加している．中でも，DoS 攻撃は 2015 年度 Q3 の被害数と 2016 年度 Q3 の被害数を見比べても 71 パーセント増加しており [1]，サービス障害を引き起こすような攻撃がさらに増えていくと予測される．DoS 攻撃増加の背景として，技術レベル・機器を問わず誰でも実行できる DoS 攻撃ツールの増加が挙げられる．DoS 攻撃は，送信元 IP アドレスを偽装（IP Spoofing）した上で行われることもあり，対策が難しい原因となっている．また，ボットネットを構築するマルウェアに感染した端末等複数の端末からの DoS 攻撃は分散型 DoS 攻撃（DDoS 攻撃）と呼ばれており，近年ではこの DDoS 攻撃が増加傾向にある．ボットネットに加担させられたホストは，知らない間に他ホストへの攻撃に加担し続けるため，思わぬ責任を負う可能性がある．IPA による 2015 年度情報セキュリティの脅威に対する意識調査 [2] では,「あなたは，次にあげるインターネット上の攻撃や被害を，どの程度脅威に感じていますか．あてはまるものをそれぞれ 1 つずつ選択してください.」という質問項目に対して，約 60.5 パーセントの回答者が「ウィルスに感染して，知らぬ間に他人のパソコンを攻撃してしまうこと」を脅威として挙げた．このことから，一般的にもユーザの知らない間に攻撃に加担することが問題であると認識されていることがわかる．

本研究では，攻撃ノードになりうる全ユーザノードを対象にし，ホストが送ろうとしているパケットをルータ内のファイアウォールを用いて制御することにより，DoS 攻撃や DDoS 攻撃への加担を防ぐための手法についての研究を行った．結果，攻撃ノードのパケットを検知し，通信を遮断することに成功した．

第1章　DoS攻撃

DoS とは “Denial of Service”の略であり，日本語ではサービス妨害と訳すことができる．DoS 攻撃とはサーバが提供する標的サービスを妨害したり，停止させたりする攻撃の総称である．

1.1　DoS 攻撃の手法

DoS 攻撃の手法は多種多岐にわたる．主な攻撃を挙げると次のようになる．

- SYN Flood
- Ping of Death
- TearDrop
- Land
- Slow Read DoS
- DNS Amp
- NTP Reflection

DoS 攻撃は大きく分けると，サービスに過負荷をかけ麻痺させる攻撃と，サービスの脆弱性を突く攻撃に分けることができる．また，OSI 基本参照モデルでの第 7 レイヤでパケットを送信する高レイヤ型攻撃と，第 2 から第 4 レイヤでパケットを送信する低レイヤ型攻撃にも分けることができる．DNS Amp 攻撃や NTP Reflection 攻撃などは，攻撃する際に踏み台となる DNS サーバや NTP サーバを見つけるまたは用意する必要がある．本研究では，Flood 攻撃と呼ばれる多数のパケットを送信することによりサービスに負荷をかけるタイプの攻撃を主な対策対象とする．

1.2　IP Spoofing

IP Spoofing とは，送信元 IP アドレスを偽装することである．元々は，攻撃対象のネットワークの内部ホストになりすまし，機密情報を手に入れるために行われていたものだが，DoS

攻撃にも利用できることが認知され，用いられるようになった．DoS 攻撃に IP Spoofing を用いる理由として，送信元 IP アドレスを偽装することによって攻撃者の身元を秘匿できるということと，送信元 IP アドレスを攻撃対象ホストの IP アドレスにすることによってリフレクタ攻撃を行うことができるということにある．前節で述べた DNS Amp 攻撃や NTP Reflection 攻撃は，このリフレクタ攻撃の一種である．図 1.1 に，リフレクタ攻撃の例を示す．

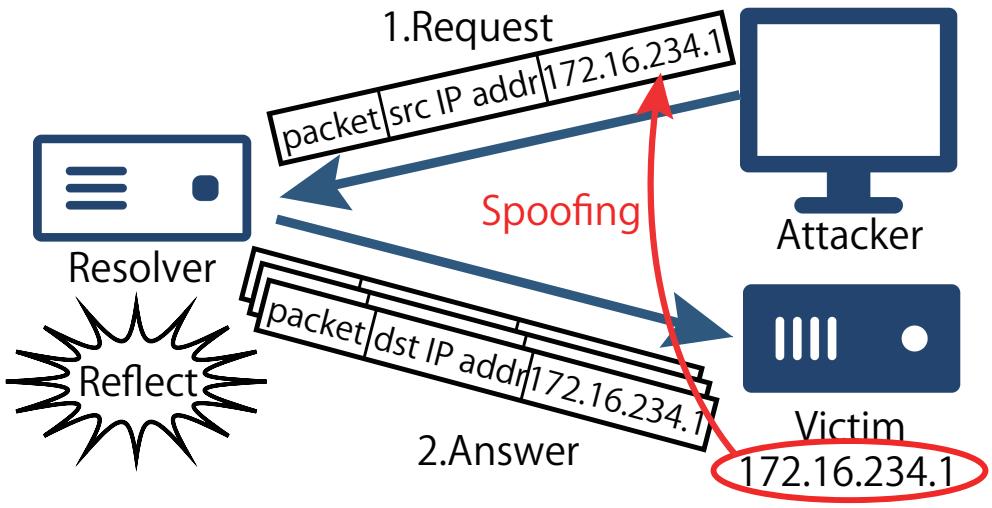

図 1.1: リフレクタ攻撃の例

1.3　DDoS 攻撃

分散型 DoS 攻撃のことを，DDoS（Distributed Denial of Service）攻撃と呼ぶ．DDoS 攻撃では複数ホストから単体の攻撃対象ホストに対して攻撃を行う．攻撃者が複数人存在する場合や，ボットネットを構築するマルウェアに感染したノードが，攻撃者の指示によって一斉に攻撃加担する場合や，攻撃者が多数のノードを所持しているか偽装している場合がある．攻撃元が分散することにより攻撃者を特定することが難しく，また攻撃的でないユーザが一斉にサーバにアクセスしている状況と区別がつかないため，対策が困難となる．

1.4　スクリプトキディ

他人が作成したツールを悪用して，第三者に攻撃を行う攻撃者のことをスクリプトキディと呼ぶ．技術的な知識がなく，自分で攻撃ツールを作成することは困難だが，ツールを入手・購入することによって攻撃を行うことが可能になる．DoS 攻撃は，攻撃ツールが簡単に入手できることや，クラウド化した DDoS 攻撃代行サービス（DDoS as a Service と呼ばれる）によってツールを所持する必要もないなど，誰でも実行することが可能な攻撃となっている．ま

た，F5 アタックと呼ばれるような手動で Web ページに GET リクエストを送りつづける手段もある．これにより，DoS 攻撃を行うスクリプトキディの数は多い．

第2章　ファイアウォール

ファイアウォールとは，異なるネットワーク間のアクセスを制限する要素または要素の集合のことである．一般に，信頼できるネットワークと信頼できないネットワーク間で通信を行う際に用いられる．ソフトウェア型とハードウェア型が存在し，本研究ではソフトウェア型のファイアウォールがデフォルトゲートウェイであるルータにインストールされていることを仮定している．OSI 参照モデルにおけるレイヤ 3・レイヤ 4 の通信の可否を判断するパケットフィルタリング型と，レイヤ 7 でパケットを精査することのできるアプリケーションゲートウェイ型があり，本研究ではパケットフィルタリング型について取り上げる．

2.1　iptables

iptables とは，Linux カーネルに組み込まれているカーネルモジュール Netfilter を操作するためのソフトウェアである．Linux 系 OS には標準でインストールされており，iptables を設定することによって，ホストをファイアウォールやルータとして動作させることができる．iptables で実装できるファイアウォールはパケットフィルタリング型で，iptables が実装されたホストが受信（INPUT）・送信（OUTPUT）・転送（FORWARD）するパケットを四つのテーブルとテーブルそれぞれのチェインによって操作する．テーブル・チェインの流れを図 2.1 に示す．それぞれのテーブル・チェインに意味があり，用途に合わせてルールを設定する．filter テーブルは INPUT・OUTPUT・FORWARD チェインを，nat テーブルは POSTROUTING・PREROUTING・OUTPUT チェインを，mangle テーブルは POSTROUTING・PREROUTING・INPUT・OUTPUT・FORWARD チェインを，raw テーブルは PREROUTING・OUTPUT チェインを持つ．また，それぞれのチェインに設定する形でユーザ定義チェインを作ることも可能である．本研究では，主に filter テーブルと filter テーブル内の FORWARD チェインを用いた．

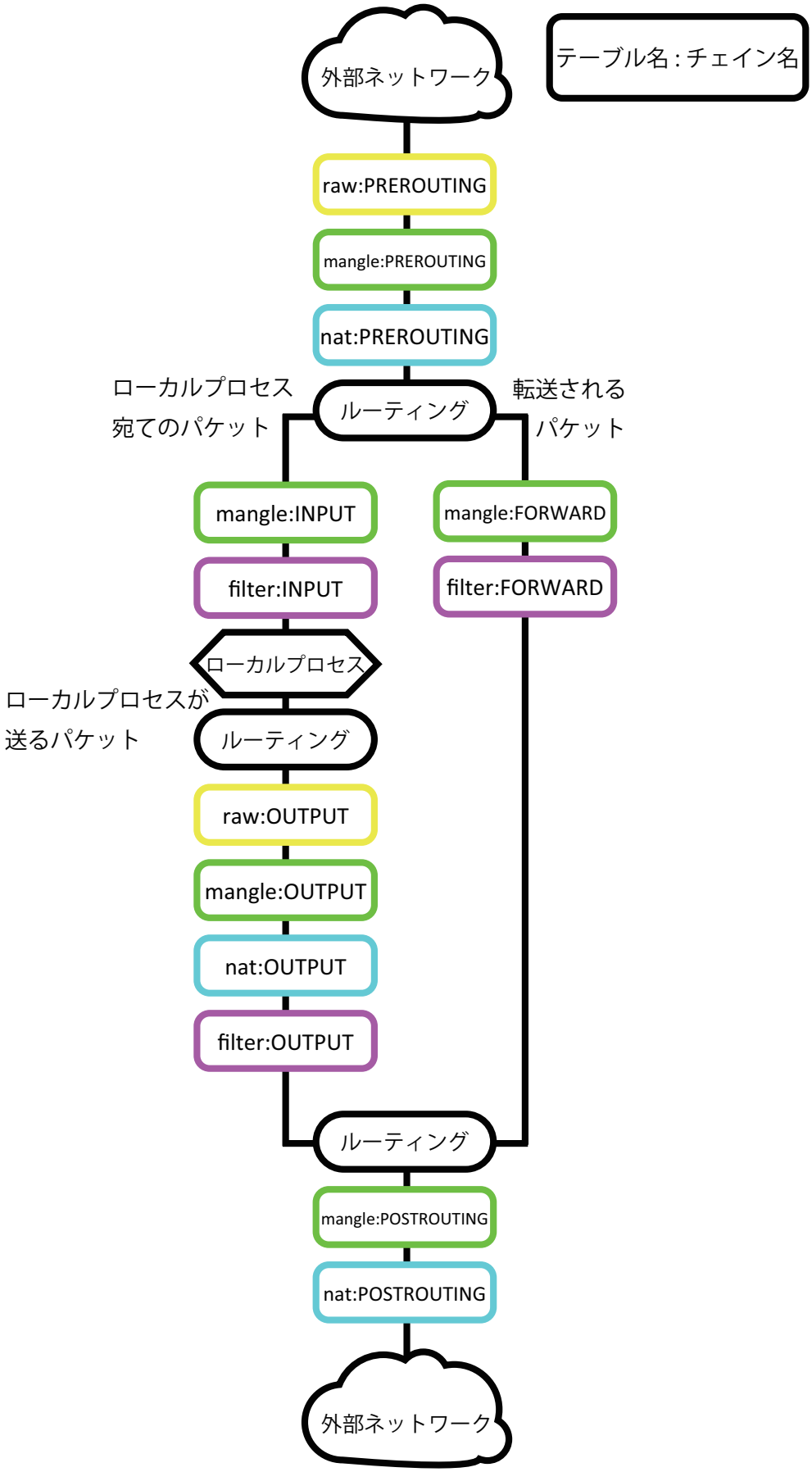

図 2.1: パケットが通る iptables のチェインの流れ

第3章　関連システム・関連研究

本章では，既存のDoS攻撃対策システムや関連する研究について取り上げる．DoS攻撃やDDoS攻撃の対策手法として，主にファイアウォールによるフィルタリングや負荷分散システムの適用が考えられるが，その中からCDNと池渕の研究について記述する．

3.1　CDN

CDNとは，Content Delivery Networkの略で，負荷分散システムの一種である．配信元となるWebサーバと，世界各地に点在し，配信元Webサーバのキャッシュを置くエッジサーバを用意する．ユーザノードからWebサイトへのアクセスがあった際に，ユーザノードそれぞれに適したエッジサーバにアクセスを分散することで，配信元Webサーバへの負荷を低減することができる．Webサーバの管理者自身がCDNの仕組みを構築するのは困難であるため，現在多くのCDNサービスが展開されている．中でも，2016年9月に発生した米国のセキュリティ情報サイト「Krebs on Security」へのDDoS攻撃[3]の際にも用いられていたAkamai社のCDNは有名である．

3.2　池渕の研究

池渕の研究[4]では，DoS攻撃の一種であるSYN Flood攻撃を，Webサーバのアクセスログに基づいた動的なパケットフィルタリングで防ぐ手法を提案した．SYN Flood攻撃とは，TCPにおける通信確立手法である3ウェイ・ハンドシェイクを悪用した攻撃である．SYN Flood攻撃では，送信元IPアドレスを偽装したSYNパケットを攻撃対象ホストに送るか，攻撃対象ホストがSYNパケットに対して返すSYN/ACKパケットに応答しないことによってhalf-open状態の通信をバッファメモリに蓄積させ，攻撃対象ホストのメモリ領域を圧迫する．池渕の研究では攻撃対象ホストをWebサーバとし，攻撃対象ホストのネットワーク上のファイアウォールで外部ホストから攻撃対象ホストに送られてくるSYNパケットの流量を制限する．流量制限の基準として，攻撃対象ホストのWebアクセスログからrate値とburst値を算出するこ

とによってトークンバケツフィルタを生成している．burst 値はバケツに溜まるトークンの最大値に，rate 値は一定時間におけるバケツへのトークンの追加量とし，ファイアウォールに iptables の limit モジュールとして適用する．パケットがファイアウォールを通過するたびにバケツからトークンを 1 ずつ減らし，バケツにトークンがなくなれば次のトークンが溜まるまでパケットの通過を制限する．burst 値は直近のアクセスログから 1 秒間の最大アクセス数を用い，rate 値は直近のアクセスログから 1 日のアクセス数の平均値を算出し，それを 1 日の秒数である 86400 で割ることにより求める．

第4章　本研究のシステム

本研究では，攻撃ノードになりうる全ユーザノードが所属するネットワークのルータを対象として，ファイアウォールによってDoS攻撃と思しき通信を遅延・遮断させるシステムを構築した．これにより，ユーザの攻撃意思に関わらず，ノードがDoS攻撃・DDoS攻撃に加担することを防ぐ．正常な通信と攻撃的な通信を見分ける手段として，正常通信時のフォワーディングパケットのログを利用して閾値を設定する．また，過検知を考慮して閾値を2段階に設定した．1段階目の閾値はパケットの遅延に，2段階目の閾値はノード自体の通信遮断に用いることで，正常な通信・ノードになるべく影響を与えず，攻撃的な通信・ノードを遮断することを目指す．

4.1　システム概要

本システムが動作する環境を図4.1に示す．攻撃ノードになりうる全ユーザノードが所属するネットワークは，一般的なユーザが自宅で利用するネットワークを想定しており，ユーザノードとゲートウェイとするルータの間にルータやプロキシを設置することは無いものとする．ルータのOSはLinux系OSを想定しており，iptablesが利用できる状態かつPython製のプログラムを動作させることができるものとする．なお，本稿執筆時に作成したシステムで用いたソフトウェアのバージョン一覧を表4.1に示す．

表4.1: ゲートウェイが用いる各システムの情報

システム	ソフトウェア名・バージョン
OS	Ubuntu14.04.05 LTS
プログラミング言語	Python 2.7.6
ファイアウォール	iptables 1.4.21

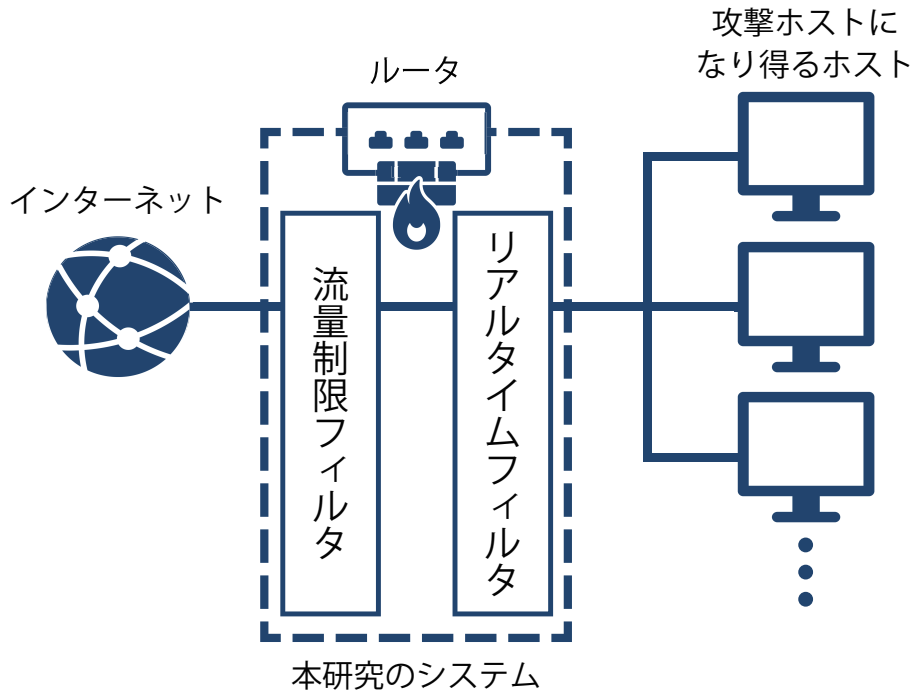

図 4.1: 本システムが動作する環境

4.2　ログの取得方法

iptables の filter テーブル FORWARD チェインにログモジュールを設定することにより，フォワーディングパケットの詳細をカーネルログに記録する．ログモジュールを設定するコマンドをリスト 4.1 に示す．-i オプションに eth1 を設定することによって eth1 の側から入ってくるパケット（内部ネットワークから入ってくるパケット）に限定してルールを適用することができる．log-prefix オプションは該当のログに特定の文字列を含ませることのできるオプションである．log-level オプションは，ログのプライオリティを設定するオプションで，ここでは info レベルとしている．log-tcp-sequence は TCP のシーケンスナンバーを記録するオプションである．log-tcp-options オプションと log-ip-options オプションは，それぞれパケットの TCP ヘッダ，IP ヘッダのオプションを記録するオプションである．このコマンドを実行することにより，/var/log/kern.log に該当パケットのログが記録されるようになる．

リスト 4.1: ゲートウェイにログモジュールを設定するコマンド

```
# iptables -t filter -A FORWARD -i eth1 -j LOG --log-prefix "
  FORWARD_F " --log-level 6 --log-tcp-sequence --log-tcp-options
  --log-ip-options
```

4.3　ログからの閾値生成

前節のログから，曜日・時間ごとに同一宛先 IP アドレスの通信と同一プロトコル・宛先ポート番号の通信をそれぞれ集計し，csv 形式で記録しておく．リスト 4.2 は，ログ集計後の csv ファイルの一例である．カラムは id，曜日番号，時間，送信元 MAC アドレス，カウントタイプ，宛先 IP アドレスまたは宛先ポート番号，集計値の順に並んでいる．曜日番号とは，それぞれの曜日を数字で表したもので，本研究では月曜日を 0 とし，日曜日を 6 とする．カウントタイプは，ログの何のデータについて集計したかを表す．それぞれのカウントタイプの意味について表 4.2 に示す．リスト 4.2 の例では，ある日曜日の 17 時のデータを参照している．この集計後のログから，同じ曜日・時間・送信元 MAC アドレス・宛先 IP アドレスまたはプロトコル・ポート番号のパケット数を集計し，その値の q パーセント点を閾値にする．実験の際にどの q が最適であるか判断する．

表 4.2: カウントタイプの意味

カウントタイプ	意味
0	宛先 IP アドレスが同じものの集計
1	ICMP パケットの集計
2	UDP プロトコルのパケットの集計
3	TCP プロトコルのパケットの集計

リスト 4.2: 集計後のログを記録する csv ファイル

```
1  278,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,104.244.43.199,428
2  279,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,104.244.42.193,2262
3  280,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,10.1.3.21,376
4  281,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,133.237.48.212,619
5  282,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,117.18.237.70,887
6  283,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,104.244.43.231,153
7  284,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,52.24.240.17,14
8  285,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,192.229.237.96,511
9  286,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,216.58.197.2,41
10 287,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,216.58.199.238,142
11 288,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,104.244.43.135,151
12 289,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,104.244.43.71,101
13 290,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,104.244.43.167,126
```

```
14 291,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,104.244.43.103,32
15 292,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,104.244.43.7,170
16 430,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,443,5018
17 431,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,2,53,376
18 432,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,80,619
```

4.3.1　閾値を用いた各種値の生成

本システムでファイアウォールを用いたフィルタリングを実施するにあたって，ログデータを利用した様々な値を用いる．記録された全ての閾値を n とすると，n の中で同じ曜日・時間・送信元 MAC アドレス・宛先 IP アドレスを持つものは n_{dst_ip} となり，同じ曜日・時間・プロトコル・宛先ポート番号を持つものは n_{dst_port} となる．n_{dst_ip} の個数を IP とし，n_{dst_port} の個数を PORT とする．i と j は 1 つずつ値を増やしていき，それぞれ，IP・PORT の値まで増えるものとする．$n_{dst\_ip_i}$ と $n_{dst\_port_j}$ を抽出し，最大値を算出して $n_{dst\_ip_i}^{day\_max}$ または $n_{dst\_port_j}^{day\_max}$ とする．また，$n_{dst\_ip_i}^{day\_max}$ の中で最大の値を n^{ip_max} とする．

$$n_{dst\_ip_i} = \{n| \text{曜日・時間・送信元 } MAC \text{ アドレス・宛先 } IP \text{ アドレスが同じ }\}.$$

$$n_{dst\_port_j} = \{n| \text{曜日・時間・送信元 } MAC \text{ アドレス・プロトコル・宛先ポート番号が同じ }\}.$$

$$n_{dst\_ip_i}^{day\_max} = max(\{n_{dst\_ip_i}\}).$$

$$n_{dst\_port_j}^{day\_max} = max(\{n_{dst\_port_j}\}).$$

$$n^{ip\_max} = max(\{n_{dst\_ip_i}^{day\_max}\}).$$

$$i = 1, 2, \cdots, IP.$$

$$j = 1, 2, \cdots, PORT.$$

4.4　本研究でのファイアウォールの構成

本研究では，iptables を用いてファイアウォールを形成する．毎時生成される速度制限のためのフィルタリングルールと，リアルタイムで生成される閾値突破時のフィルタリングルールがある．それぞれ，送信元 MAC アドレスを用いて送信元ノードを区別し，フィルタリングする．

4.4.1　流量制限のためのフィルタリング

フォワーディングパケットに正常な通信のみあるのか，攻撃的な通信が混ざっているのか判別がつかないときに，パケットの流量制限を行うフィルタリングルールが適用される．毎時，閾値と iptables の limit モジュールを用いて，池渕の研究 [4] にあるようなトークンバケツフィルタを生成する．本研究での rate 値と burst 値は，検知対象をパケットの宛先 IP アドレスとするか宛先ポートとするかによって別の値を用いる．検知対象が宛先ポート番号である場合，$n^{day\_max}_{dst\_port_j}$ を 1 時間の秒数である 3600 で割ったものを rate 値とする．burst 値はその曜日・時間台の閾値となる．検知対象を宛先 IP アドレスとする場合，n^{ip_max} を 1 時間の秒数である 3600 で割ったものを rate 値とする．burst 値は n^{ip_max} とする．また，rate 値は 1 を下回る値を取れないため，算出の結果 1 を下回ることになれば 1 を設定する．burst 値は 10000 を上回る値を取れないため，閾値が 10000 を上回る場合は 10000 を設定する．プロトコルが TCP で，宛先ポート番号が 80 番のパケットのフィルタリング例をリスト 4.3 に示す．host0 とは，後述の applying.py で設定される送信元 MAC アドレスごとの独自チェイン名である．

$$rate(dst\_port) = \begin{cases} 1, & (n^{day\_max}_{dst\_port_j}/3600 < 1) \\ n^{day\_max}_{dst\_port_j}/3600. & (otherwise) \end{cases}$$

$$burst(dst\_port) = \begin{cases} 10000, & (\text{曜日・時間の閾値} > 10000) \\ \text{曜日・時間の閾値}. & (otherwise) \end{cases}$$

$$rate(dst\_ip) = \begin{cases} 1, & (n^{ip\_max}/3600 < 1) \\ n^{ip\_max}/3600. & (otherwise) \end{cases}$$

$$burst(dst\_ip) = \begin{cases} 10000, & (n^{ip\_max} > 10000) \\ n^{ip\_max}. & (otherwise) \end{cases}$$

リスト 4.3: TCP のポート 80 番への流量制限フィルタリング例

```
# iptables -t filter -A host0 -p tcp --sport 80 -m limit -limit
   52/second --limit-burst 10000
```

4.4.2　limit 値を超えた場合のリアルタイムフィルタリング

攻撃的な通信がある確率が高い場合に，攻撃的な通信を行っているノードの通信を禁止するフィルタリングルールが適用される．1 秒間の通信の中で，同一送信元ノード・宛先 IP アドレスまたはプロトコルと宛先ポート番号のパケットを集計する．1 秒の間に limit 値を超える数のパケットを検知すると，その送信元ノードの通信を禁止する．後述の application.csv に格納されている同一曜日・時間・送信元 MAC アドレス・宛先 IP アドレスまたはプロトコルと宛先ポート番号の閾値を，limit 値で扱う同一情報とする．同一情報を 1 時間の分数である 60 で割った値を limit 値とする．また，limit 値が 1 を下回る場合は 1 を limit 値とする．1 秒間に 1 分相当の同一情報を持つパケットを検知した場合に，送信したノードが外部通信を禁止されるようになる．リスト 4.4 は，ノードの通信を禁止するルールの一例である．

$$limit = \begin{cases} 1, & (\text{同一情報の閾値}/60 < 1) \\ \text{同一情報の閾値}/60. & (otherwise) \end{cases}$$

リスト 4.4: ノードの通信を禁止するフィルタリング例

```
# iptables -I FORWARD 2 -m mac --mac-source 00:50:56:b9:50:67 -j
   banned
```

banned とは，通信を禁止したいノードの通信が転送される独自チェインである．banned に設定されたルールをリスト 4.5 に示す．alert レベルでカーネルログにパケットの詳細を記録した後，パケットを破棄する．

リスト 4.5: banned チェインのルール

```
1 # iptables -t filter -A banned -j LOG --log-prefix "
     BANNED_USER_PACKET " --log-level alert --log-tcp-sequence --log
     -tcp-options --log-ip-options
2 # iptables -t filter -A banned -j DROP
```

4.5　プログラム

ここでは本システムで利用するプログラムについて取り上げる．log_formatting.py，log_counter.py，calculating_threshold.py，comparing_threshold.py は毎日 0 時 0 分に，applying.py は毎時 0 分に実行する．packet_counter.py は常時実行される．

4.5.1 log_formatting.py

log_formatting.py は，カーネルログに記録された前日分のフォワーディングパケットのログを後述の log_counter.py で数えやすい csv 形式に整形するプログラムである．log_formatting.py は毎日 0 時 0 分に実行される．引数として，カーネルログを与える．リスト 4.6 は整形前のログを，リスト 4.7 は整形後のログを示す．

リスト 4.6: 整形前のログ

```
1 Jan 24 17:44:31 k196336-router-ubuntuserver kernel:
    [592535.171421] FORWARD_F IN=eth1 OUT=eth0 MAC=00:50:56:b9:e8:
    a2:00:50:56:b9:50:67:08:00 SRC=192.168.1.11 DST=10.1.167.31 LEN
    =40 TOS=0x00 PREC=0x00 TTL=63 ID=256 PROTO=TCP SPT=1073 DPT=80
    SEQ=1344664235 ACK=0 WINDOW=8192 RES=0x00 SYN URGP=0
2 Jan 24 17:46:23 k196336-router-ubuntuserver kernel:
    [592647.681487] FORWARD_F IN=eth1 OUT=eth0 MAC=00:50:56:b9:e8:
    a2:00:50:56:b9:50:67:08:00 SRC=192.168.1.11 DST=10.1.3.1 LEN=84
     TOS=0x00 PREC=0x00 TTL=63 ID=48309 DF PROTO=ICMP TYPE=8 CODE=0
     ID=12909 SEQ=1
```

リスト 4.7: 整形後のログ

```
1 246639,1,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,10.1.167.31,TCP,80
2 246640,1,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,10.1.3.1,ICMP,0
```

整形後のログは formatted.csv として保存される．カラムはそれぞれ，その日の通信の何パケット目のパケットであるかを利用した ID 値と，その日の曜日情報（0 が月曜日となり，6 が日曜日となる）と，そのパケットが何時台のパケットであるかと，送信元 MAC アドレス，宛先 IP アドレス，プロトコル，ポート番号である．本研究で使用されるログに記録される MAC アドレスだが，宛先 MAC アドレス（本研究ではゲートウェイの MAC アドレス）と送信元 MAC アドレスと Ethernet frame のタイプが結合したものとなっている [5]．フィルタリング時には送信元 MAC アドレスのみ切り出すが，ログ記録時は切り出さずに利用している．また，ICMP のポート番号情報は無いため，ポート番号のカラムは 0 としておく．

4.5.2　log_counter.py

log_counter.py は，前述の log_formatting.py で整形されたログファイル formatted.csv から，同一時間･同一送信元 MAC アドレス，同一宛先 IP アドレスまたは同一プロトコル･宛先ポート番号のものを集計するプログラムである．log_counter.py は毎日 0 時 0 分に log_formatting.py の後に実行される．集計したものは，counted.csv として逐次保存される．counted.csv の例をリスト 4.8 に載せる．カラムはそれぞれ，ID，曜日情報，そのパケットが送られた時間台，送信元 MAC アドレス，カウントタイプ，宛先 IP アドレスまたは宛先ポート番号，集計された値，となる．カウントタイプとは，集計値が何の集計値であるかを表し，0 ならば宛先 IP アドレス，1 ならば ICMP プロトコル，2 ならば UDP プロトコル，3 ならば TCP プロトコルを表す．リスト 4.8 の例では，1 行目はある日曜日の 17 時台の MAC アドレス 00:50:56:b9:50:67 のノードから 104.244.43 宛てに送られたパケットが 32 件あったことを表している．

リスト 4.8: 集計後のログ

```
1 291,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,104.244.43.103,32
2 292,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,104.244.43.7,170
3 430,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,443,5018
4 431,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,2,53,376
5 432,6,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,80,619
```

4.5.3　calculating_threshold.py

calculating_threshold.py では，前述の log_counter.py が集計したログ counted.csv から，本日分の閾値を計算する．calculating_threshold.py は毎日 0 時 0 分に実行される．計算された閾値は application.csv として保存される．counted.csv から，本日と同一曜日のログを抜き出し，同一時間台・送信元 MAC アドレス・カウントタイプ・宛先 IP アドレスまたは宛先ポートの集計値から四分位値を算出し，それを閾値として保存する．リスト 4.9 に application.csv のデータ例を挙げる．また例として，application.csv にリスト 4.10 のようなログがあった場合の閾値を算出する．

リスト 4.9: application.csv の例

```
1 135,1,15,00:50:56:b9:e8:a2:00:50:56:b9:92:9c:08:00,2,53,2
2 136,1,15,00:50:56:b9:e8:a2:00:50:56:b9:92:9c:08:00,3,80,5
3 137,1,16,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,10.1.3.21,25
4 138,1,16,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,10.1.3.80,4
5 141,1,16,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,2,53,29
```

```
142,1,16,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,443,87
143,1,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,10.1.3.21,13
144,1,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,10.1.3.80,6
147,1,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,2,53,19
148,1,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,443,86
150,1,17,00:50:56:b9:e8:a2:00:50:56:b9:92:9c:08:00,0,10.1.3.80,15
155,1,17,00:50:56:b9:e8:a2:00:50:56:b9:92:9c:08:00,2,53,15
156,1,17,00:50:56:b9:e8:a2:00:50:56:b9:92:9c:08:00,3,443,170
157,1,17,00:50:56:b9:e8:a2:00:50:56:b9:92:9c:08:00,3,80,9186
```

リスト 4.10: 同一曜日・時間台・送信元 MAC アドレス・プロトコル・ポート番号のデータ例

```
32,0,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,80,5
22,0,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,80,9186
661,0,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,80,18726
432,0,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,80,619
434,0,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,80,624
665,0,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,80,764
29,0,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,80,5
169,0,17,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,80,13847
```

全てのデータの最終カラムをソートすると，$5, 5, 619, 624, 764, 9186, 13847, 18726$ となる．q の値が 75 であった場合，このデータの閾値は 10352 となる．

4.5.4　comparing_threshold.py

comparing_threshold.py は，前述の calculating_threshold.py が算出した閾値が保存されている application.csv から，同一送信元 MAC アドレス・宛先 IP アドレスまたはプロトコルと宛先ポートのデータを取り出し，その中で最大の閾値を算出して most_threshold.csv に格納する．リスト 4.11 に，most_threshold.csv の例を示す．

リスト 4.11: most_threshold.csv の例

```
0,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,10.1.167.31,658
1,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,10.1.3.21,340
```

```
2,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,0,10.1.3.80,6
58,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,2,53,340
59,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,443,1135
60,00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00,3,80,5805
61,00:50:56:b9:e8:a2:00:50:56:b9:92:9c:08:00,0,10.1.167.31,10462
62,00:50:56:b9:e8:a2:00:50:56:b9:92:9c:08:00,0,10.1.3.21,32
63,00:50:56:b9:e8:a2:00:50:56:b9:92:9c:08:00,0,10.1.3.80,18
75,00:50:56:b9:e8:a2:00:50:56:b9:92:9c:08:00,2,53,34
76,00:50:56:b9:e8:a2:00:50:56:b9:92:9c:08:00,3,443,170
77,00:50:56:b9:e8:a2:00:50:56:b9:92:9c:08:00,3,80,10467
78,00:50:56:b9:e8:a2:00:50:56:b9:f0:d6:08:00,0,10.1.3.80,4
86,00:50:56:b9:e8:a2:00:50:56:b9:f0:d6:08:00,2,53,4
```

4.5.5　applying.py

applying.py は，前述の calculating_threshold.py と comparing_threshold.py から生成された application.csv と most_threshold.csv を用いて，iptables に流量制限のフィルタリングルールを適用する applying_iptables.sh を生成する．applying.py は毎時 0 分に実行される．applying_iptables.sh の例をリスト 4.12 に示す．

リスト 4.12: applying_iptables.sh の例

```
iptables -D FORWARD 2
iptables -F host0
iptables -X host0
iptables -D FORWARD 2
iptables -F host1
iptables -X host1
iptables -N host0
iptables -A FORWARD -m mac --mac-source 00:50:56:b9:50:67 -j host0
iptables -A host0 -p udp --sport 53 -m limit --limit 1/second --
   limit-burst 15
iptables -A host0 -p tcp --sport 443 -m limit --limit 1/second --
```

```
    limit-burst 85
11 iptables -N host1
12 iptables -A FORWARD -m mac --mac-source 00:50:56:b9:92:9c -j host1
13 iptables -A host1 -p tcp --sport 443 -m limit --limit 1/second --
    limit-burst 27
14 iptables -A host1 -p tcp --sport 80 -m limit --limit 38/second --
    limit-burst 15
15 iptables -A host0 -m hashlimit --hashlimit 1/sec --hashlimit-mode
    dstip --hashlimit-name sameip_filter
16 iptables -A host1 -m hashlimit --hashlimit 1/sec --hashlimit-mode
    dstip --hashlimit-name sameip_filter
17 iptables-save > /etc/iptables/iptables.rules
```

1 から 6 行目は，前時間に設定されたノードごとの設定を消去する設定である．7 から 10 行目と 11 行目から 14 行目は，それぞれノードごとの設定を行っている．7 行目と 11 行目でノードごとの独自チェインを作成し，8 行目と 12 行目でどの送信元 MAC アドレスを持つノードのパケットをそれぞれの独自チェインに転送するかを設定している．9 行目・10 行目・13 行目・14 行目は，それぞれプロトコル・宛先ポート番号ごとのトークンバケツフィルタを設定している．9 行目を例に取ると，1/second という値が rate 値になり，15 という値が burst 値となる．rate 値は most_threshold.csv から同一送信元 MAC アドレス・プロトコル・宛先ポート番号の閾値を抽出し，1 秒間の秒数である 3600 で割ったものになる．burst 値は，applicatoin.csv から同一時間台・送信元 MAC アドレス・プロトコル・宛先ポート番号の閾値を抽出する．15 行目・16 行目は，それぞれノードごとに hashlimit によってトークンバケツフィルタを設定している．ここでのフィルタは同一宛先 IP アドレスごとにハッシュテーブルに値を追加していく．これにより，同一宛先 IP アドレスを持つパケットを，それぞれの宛先 IP アドレスごとに同じ rate 値と burst 値を用いて流量制限をすることができる．15 行目を例に取ると，1/sec が rate 値になり，burst 値にはデフォルトの 5 が入っている．rate 値は most_threshold.csv から宛先 IP アドレスを対象とした閾値の中で最大の値を 3600 で割ったものとなり，burst 値は宛先 IP アドレスを対象とした閾値の中で最大の値となる．それぞれの値は 0 という値は設定できないため，0 である場合は 1 とする．また，burst 値は 10000 以上の値を設定できないため，10000 を超える場合は 10000 を設定する．

4.5.6　packet_counter.py

packet_counter.py は，4.2.2 節の閾値突破時のパケットフィルタリングを実装するプログラムである．前述の calculating_threshold.py から生成された application.csv を用いて，limit 値を計算しながらフォワーディングパケットをリアルタイムに集計する．パケットは 1 秒ごとにカウントされており，1 秒間に同一送信元 MAC アドレス・宛先 IP アドレスまたはプロトコルと宛先ポートのものを limit 値以上集計すると，該当の宛先ノードのフォワーディングパケットを全て遮断する．packet_counter.py は Cython によってライブラリ化されており，packet_counter_pipe.py から呼び出される形で実行される．

4.6　本システムを利用する流れ

本システムは，一般的なユーザが外部ネットワークとフォワーディングパケットのやり取りをするログが必要となる．そのため，フィルタリングシステム無しで本システムの環境下にあるノードを正常に利用する期間が 1 週間以上必要である．正常利用時のログを取得する期間の流れを図 4.2 に示す．十分なログを取得した後，本システムのフィルタリング機構を適用する．本システムを適用した後の流れは，図 4.3 のようになる．図 4.2 と図 4.3 にある括弧内のプログラムは，その処理を実行する際に用いられるプログラムである．また，図 4.2 に記述されている「ログモジュールの設定」という処理だが，これは本稿 4.2 節で述べられているコマンドを用いる．同じく図 4.2 に記述されている「本システムの適用」という処理だが，これは crontab にそれぞれのプログラムを設定し，packet_counter.py に常時カーネルログを処理させるシェルスクリプトである realtime.sh を，常時実行するプロセスとして実行することを指す．

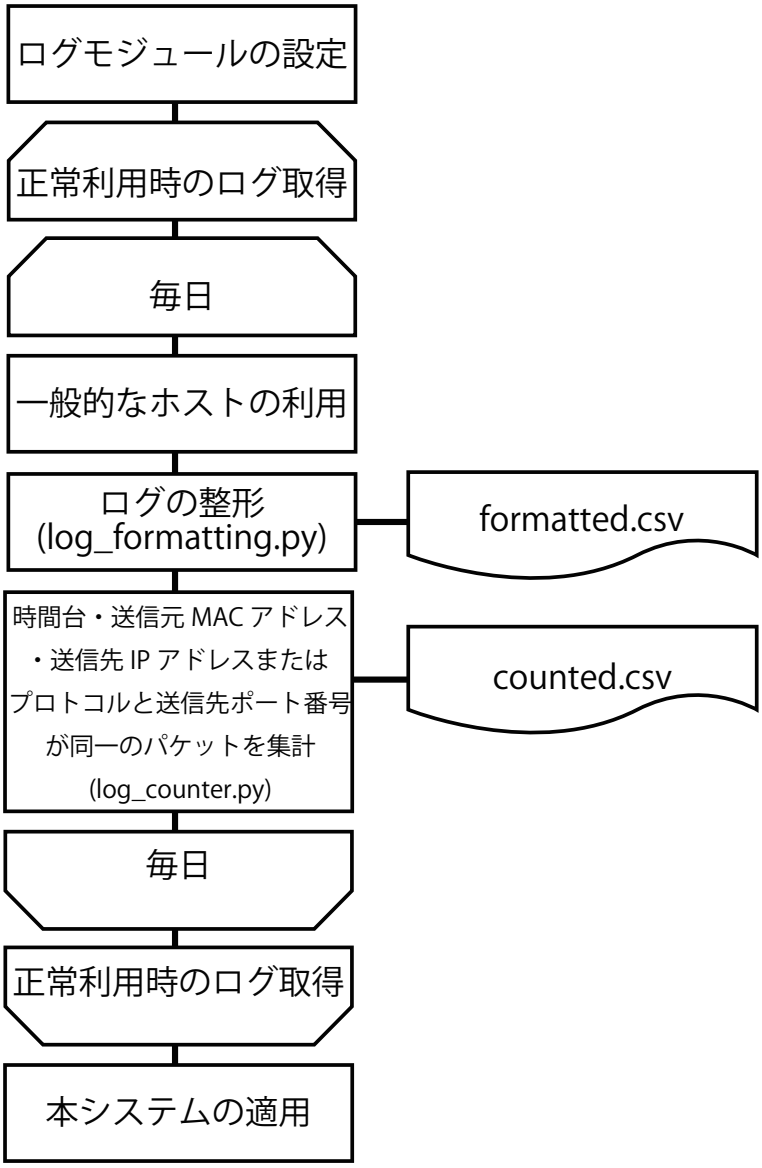

図 4.2: 正常利用時のログを取得する期間の流れ

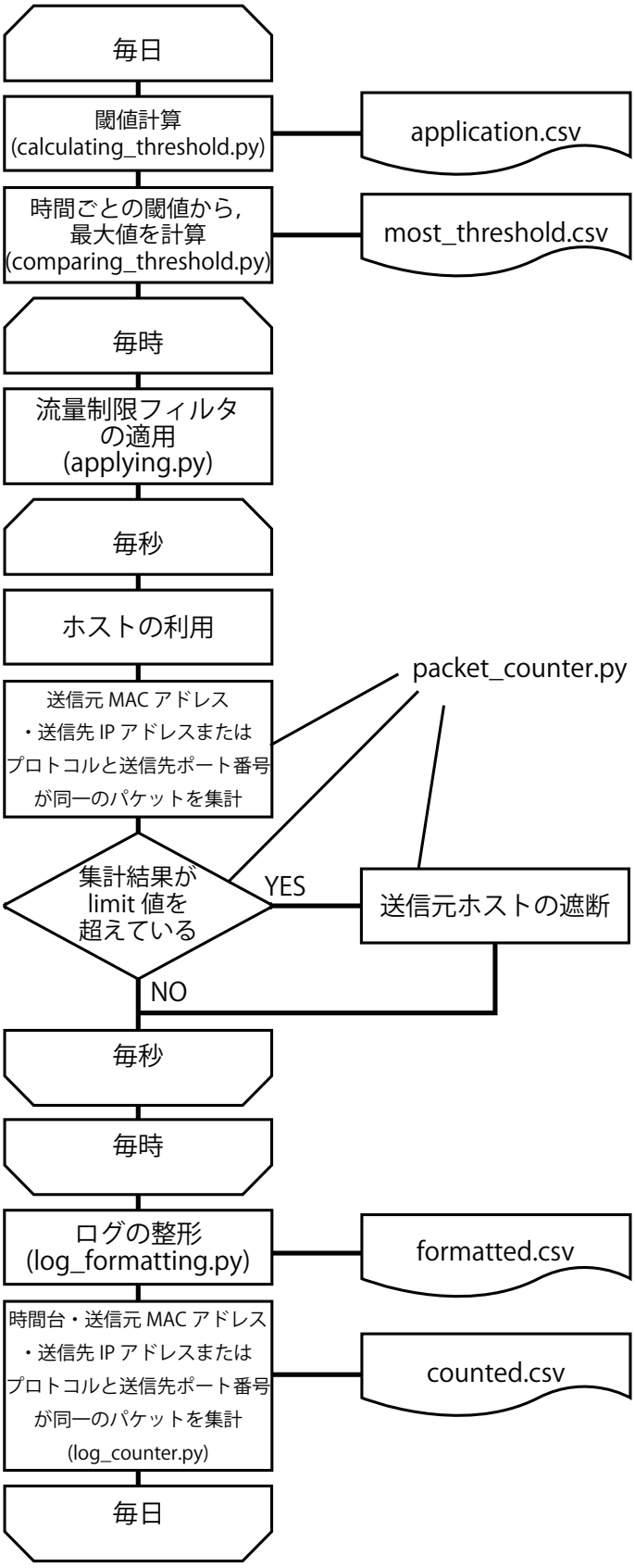

図 4.3: 本システム利用時の流れ

第5章　実験と考察

本章は，本研究のシステムを用いて実験を行った結果と，その考察を記述する．実験は，本研究のシステムが動作する環境下で，悪意あるユーザがノードを利用して外部サーバを攻撃していることを想定してシミュレーションを行う．

5.1　実験環境

実験環境は，後述のシナリオ A・シナリオ B に合わせて，図 5.1 の通りに用意した．なおこの環境は，小林研究室で運営されている VMware vSphere サーバ上に構築されている．各ノードの OS のバージョンと IP アドレスと MAC アドレスは表 5.1 の通りである．

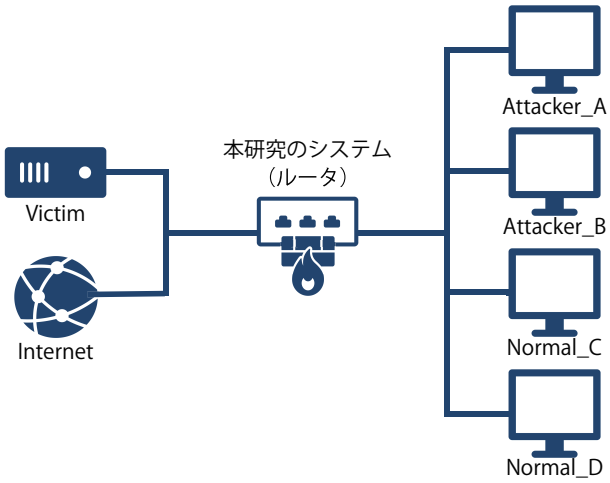

図 5.1: 実験環境

5.2　実験のシナリオ

本節では，実験に用いたシナリオについて説明する．本研究のシステムを利用したいと考えられる状況からシナリオ A，B を考案した．また，それぞれのノードの平常時のログを 3 週間分記録できているものとする．攻撃対象は Victim に限定し，それぞれのノードは Victim 宛ての正常な通信は行わないものとする．本システムが一般ユーザのノード利用に与える影響を調

表 5.1: 各ノードの OS と IP アドレス

ノード名	OS	IP アドレス	MAC アドレス
Gateway	Ubuntu Server 14.04.5 LTS	10.1.167.25	00:50:56:b9:49:48
		192.168.1.25	00:50:56:b9:e8:a2
Attacker_A	Ubuntu Server 14.04.5 LTS	192.168.1.30	00:50:56:b9:92:9c
Attacker_B	Windows 8	192.168.1.33	00:50:56:b9:b4:35
Normal_C	Lubuntu 14.04.5 LTS	192.168.1.11	00:50:56:b9:50:67
Normal_D	Ubuntu Server 14.04.5 LTS	192.168.1.32	00:50:56:b9:f0:d6
Victim	Ubuntu Server 14.04.5 LTS	10.1.167.31	00:50:56:b9:2b:6d

べるために，シナリオ A・B それぞれに一般的なユーザが利用するノードとして Normal_C・Normal_D を用意した．この 2 つのノードは正常な通信を行うが，フィルタリングの対象である．Normal_C または Normal_D の通信をフィルタリングした場合，誤検知となる．また，Normal_C は普段から活発に利用されているノードとし，Normal_D はあまり利用されていないノードとする．

5.2.1　シナリオ A - Linux で DoS 攻撃ツールが利用されている場合

Attacker_A が，正常なユーザと悪意を持ったユーザ両方に利用されている場合を想定する．正常なユーザは Victim 以外のノードにアクセスしており，悪意を持ったユーザは Victim に対して DoS 攻撃を仕掛ける．悪意を持ったユーザが利用する攻撃ツールとして，池渕の研究 [4] で用いられている tgn コマンドを利用する．リスト 5.1 は，利用した tgn コマンドである．

リスト 5.1: 実験に使用した tgn コマンド

```
1 sudo tgn -c 6000 -r 100 "ip(id = 'random', dst = "10.1.167.31")/
    tcp(syn, seq = 'random', src = 'random', dst = 80)"
```

5.2.2　シナリオ B - Windows で DoS 攻撃ツールが利用されている場合

Attacker_B が，悪意を持ったユーザによって利用されている場合を想定する．悪意を持ったユーザは Attacker_B に DoS 攻撃ツールをインストールし，Victim の 80 番ポートに攻撃を仕掛ける．この際，悪意を持ったユーザは攻撃に専念しており，その他の通信は行っていないものとする．シナリオ A と同じく，Normal_C・Normal_D はマルウェアに感染しておらず，一般のユーザが利用している．DoS 攻撃ツールとして，一般的に利用されている LOIC[7] を

用いた．

5.3　評価方法

本システムでのリアルタイムフィルタリングによりノードの通信を遮断する際に，4.4.2 節のリスト 4.5 にある banned チェインを禁止されたノードのパケットが通るよう適用している．banned チェインでは，禁止されたノードのパケットをカーネルログに記録し，その後パケットを破棄している．禁止されたノードのパケットは，一般のパケットの情報がカーネルログに記録されるルールも通過しているため，禁止されたノードのパケットのみカーネルログに二重に情報が残ることになる．一般のパケットの情報が記録されているログの数からフォワーディングパケットの総数を得て，禁止されたノードのパケットの情報が記録されているログの数をフォワーディングパケットの総数から引くと，通信を行うことができたパケットの数となる．同じパケットを，一般のパケットの情報として記録したログと禁止されたノードのパケットの情報として記録したログをリスト 5.2 に示す．同じパケットであるため，シーケンス番号は同一となる．

リスト 5.2: 一般のパケットログ（1 行目）と禁止されたノードのパケットログ（2 行目）

```
1 Jan 30 18:31:21 k196336-router-ubuntuserver kernel:
    [1113956.410949] FORWARD_F IN=eth1 OUT=eth0 MAC=00:50:56:b9:e8:
    a2:00:50:56:b9:50:67:08:00 SRC=192.168.1.11 DST=10.1.167.31 LEN
    =40 TOS=0x00 PREC=0x00 TTL=63 ID=256 PROTO=TCP SPT=1073 DPT=443
     SEQ=1255524723 ACK=0 WINDOW=8192 RES=0x00 SYN URGP=0
2 Jan 30 18:31:21 k196336-router-ubuntuserver kernel:
    [1113956.410973] BANNED_USER_PACKET IN=eth1 OUT=eth0 MAC
    =00:50:56:b9:e8:a2:00:50:56:b9:50:67:08:00 SRC=192.168.1.11 DST
    =10.1.167.31 LEN=40 TOS=0x00 PREC=0x00 TTL=63 ID=256 PROTO=TCP
    SPT=1073 DPT=443 SEQ=1255524723 ACK=0 WINDOW=8192 RES=0x00 SYN
    URGP=0
```

正常な通信の量と不正な通信の量を知るために，それぞれのノードにもログモジュールを実装する．本実験では Victim への通信は全て不正な通信とし，それ以外の通信は正常な通信とする．正常な通信のログと不正な通信のログを分けるために，リスト 5.3 の iptables のコマンドを各ノードで実行する．この際，Attacker_B は OS が Windows であるためログを取るこ

とが不可能であるが，シナリオ B で Attacker_B は不正な通信のみ行うため，ログを取る必要は無いものとする．

リスト 5.3: 各ノードへのログモジュール実装ルール

```
# iptables -N to_gateway
# iptables -A OUTPUT -d 192.168.1.25 -j to_gateway
# iptables -A to_gateway -j ACCEPT
# iptables -N badlog
# iptables -A OUTPUT -d 172.16.1.31 -j badlog
# iptables -A badlog -j LOG --log-prefix "BAD_LOG " --log-level
   alert --log-tcp-sequence --log-tcp-options --log-ip-options
# iptables -A badlog -j ACCEPT
# iptables -A OUTPUT -j LOG --log-prefix "NOMAL_LOG " --log-level
   info --log-tcp-sequence --log-tcp-options --log-ip-options
```

禁止されたノードのログから，正常な通信のログと不正な通信のログ（Victim 宛ての通信のログ）を分けることにより，誤検知を行っているかどうかが判定できる．また，通信を行うことができたパケットのログから，正常な通信のログと不正な通信のログを分けることにより，検知できていないパケットの量を知ることができる．このことから，正常な通信で正常であると判定された通信・正常な通信で不正であると判定された通信（false-positive）・不正な通信で正常であると判定された通信（false-negative）・不正な通信で不正であると判定された通信に，通信を分けることができる．false-negative の通信量を抑えつつ，false-positive の通信量をできる限り小さくすることを目標とする．正常な通信のうち，ルータが遮断したパケットがいくつあるかを誤検知率とし，不正な通信のうち，ルータが遮断できなかったパケットがいくつあるかを不検知率とする．

5.4　実験 A

5.2 節の実験のシナリオに基づき，10 分間の実験を行った．実験用のログ・閾値データとして火曜日 20 時のデータを用いた．q が 70・75・80・90・100 の場合の閾値を算出し，流量制限フィルタを適用した状態で実験を行った．q が 75 である場合の火曜日 20 時の各ノードの閾値から，Victim に関わるものを抜き出し，表 5.2 に示す．q が 75 である場合の火曜日 20 時の閾値から算出された流量制限フィルタのルールは，リスト 5.4 のようになる．図 5.1 の実験環

境において，Normal_C を host0 とし，Attacker_A を host1 とし，Attacker_B を host2 とし，Normal_D を host3 とする．q に応じた流量制限のルールを設定した上で，実験を行う．一般ユーザは最初の 5 分間はリンク数の少ないサイト A を閲覧しており，最後の 5 分間はリンク数の多いサイト B を閲覧している．悪意のあるユーザは実験開始の 5 分後に攻撃を行う．

表 5.2: q が 75 での火曜日 20 時の各ノードの閾値

ノード名	カウントタイプ	宛先 IP アドレスまたは宛先ポート番号	閾値
Attacker_A	0	172.16.1.31	6
Attacker_A	3	80	1106
Attacker_A	3	443	483
Attacker_B	0	172.16.1.31	10
Attacker_B	3	80	1342
Attacker_B	3	443	25746
Normal_C	0	172.16.1.31	37
Normal_C	3	80	13749
Normal_C	3	443	22211
Normal_D	0	172.16.1.31	6
Normal_D	1	-	3
Normal_D	3	443	749

リスト 5.4: q が 75 での流量制限フィルタ

```
1 iptables -N host0
2 iptables -A FORWARD -m mac --mac-source 00:50:56:b9:50:67 -j host0
3 iptables -A host0 -p udp --sport 123 -m limit --limit 1/second --
    limit-burst 32 -j ACCEPT
4 iptables -A host0 -p udp --sport 53 -m limit --limit 1/second --
    limit-burst 4117 -j ACCEPT
5 iptables -A host0 -p tcp --sport 443 -m limit --limit 9/second --
    limit-burst 10000 -j ACCEPT
6 iptables -A host0 -p tcp --sport 53 -m limit --limit 1/second --
    limit-burst 7 -j ACCEPT
```

```
7  iptables -A host0 -p tcp --sport 80 -m limit --limit 5/second --
     limit-burst 10000 -j ACCEPT
8  iptables -N host1
9  iptables -A FORWARD -m mac --mac-source 00:50:56:b9:92:9c -j host1
10 iptables -A host1 -p udp --sport 53 -m limit --limit 1/second --
     limit-burst 81 -j ACCEPT
11 iptables -A host1 -p tcp --sport 25 -m limit --limit 1/second --
     limit-burst 4 -j ACCEPT
12 iptables -A host1 -p tcp --sport 443 -m limit --limit 1/second --
     limit-burst 510 -j ACCEPT
13 iptables -A host1 -p tcp --sport 80 -m limit --limit 1/second --
     limit-burst 2741 -j ACCEPT
14 iptables -N host2
15 iptables -A FORWARD -m mac --mac-source 00:50:56:b9:b4:35 -j host2
16 iptables -A host2 -p udp --sport 53 -m limit --limit 1/second --
     limit-burst 1180 -j ACCEPT
17 iptables -A host2 -p tcp --sport 443 -m limit --limit 11/second --
     limit-burst 10000 -j ACCEPT
18 iptables -A host2 -p tcp --sport 80 -m limit --limit 2/second --
     limit-burst 7807 -j ACCEPT
19 iptables -N host3
20 iptables -A FORWARD -m mac --mac-source 00:50:56:b9:f0:d6 -j host3
21 iptables -A host3 -p icmp -m limit --limit 1/second --limit-burst
     3 -j ACCEPT
22 iptables -A host3 -p udp --sport 53 -m limit --limit 1/second --
     limit-burst 27 -j ACCEPT
23 iptables -A host3 -p tcp --sport 443 -m limit --limit 1/second --
     limit-burst 565 -j ACCEPT
24 iptables -A host0 -m hashlimit --hashlimit 4/sec --hashlimit-burst
     10000 --hashlimit-mode dstip --hashlimit-name sameip_filter --
```

```
   hashlimit-htable-expire 3600000 -j ACCEPT
25 iptables -A host1 -m hashlimit --hashlimit 1/sec --hashlimit-burst
    3002 --hashlimit-mode dstip --hashlimit-name sameip_filter --
   hashlimit-htable-expire 3600000 -j ACCEPT
26 iptables -A host2 -m hashlimit --hashlimit 6/sec --hashlimit-burst
    10000 --hashlimit-mode dstip --hashlimit-name sameip_filter --
   hashlimit-htable-expire 3600000 -j ACCEPT
27 iptables -A host3 -m hashlimit --hashlimit 1/sec --hashlimit-burst
    4286 --hashlimit-mode dstip --hashlimit-name sameip_filter --
   hashlimit-htable-expire 3600000 -j ACCEPT
```

5.4.1　q = 70 の場合

シナリオ A - Linux で DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 7,153 パケットである．各ノードは正常な通信を 1,153 パケット送り，Attacker_A は不正な通信を 6000 パケット送信した．その結果，本システムが 6,277 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.3 のようになった．誤検知率は 24%となり，不検知率は 0%となる．遮断されるノード数の推移を図 5.2 に示す．実験を開始して 2 分 4 秒後に Normal_D の通信を遮断し，4 分 17 秒後に Attacker_A の通信を遮断し，4 分 42 秒後に Normal_C を遮断する．各ノードの遮断時の limit 値を表 5.4 に示す.

表 5.3: q = 70 でのシナリオ A の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	876	0
ゲートウェイから遮断された通信	277	6,000

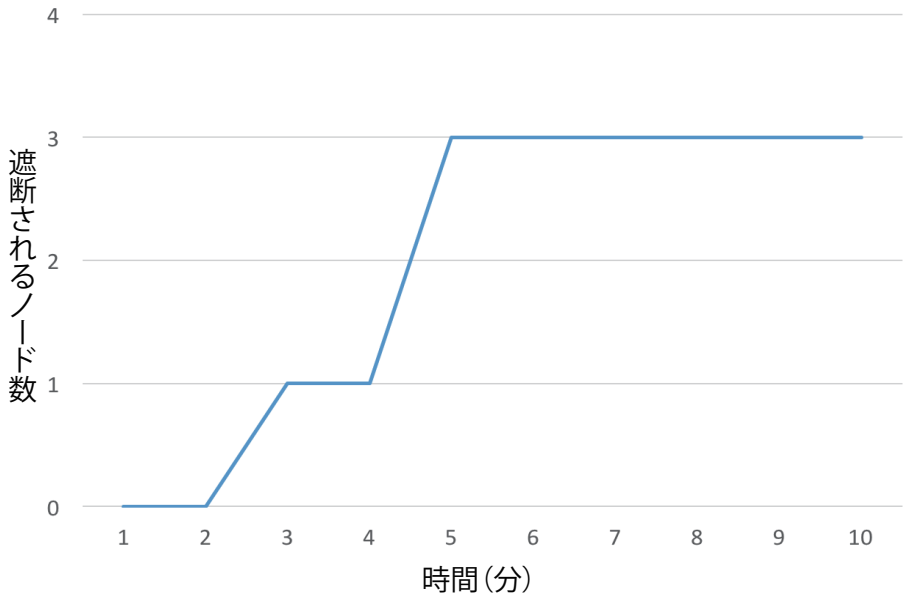

図 5.2: q = 70 でのシナリオ A の分単位での遮断されたノード数の推移

表 5.4: q = 70 でのシナリオ A の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_A	10.1.3.80	1
Normal_C	172.217.26.106	1
Normal_D	10.1.3.80	1

シナリオ B - DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 9,834 パケットである．Attacker_B を除く各ノードは正常な通信を 9,417 パケット送り，Attacker_B は不正な通信を 417 パケット送信した．その結果，本システムが 1,273 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.5 のようになった．誤検知率は 9%となり，不検知率は 0%となる．遮断されるノード数の推移を図 5.3 に示す．実験を開始して 1 分 50 秒後に Attacker_B の通信を遮断する．6 分 4 秒後に Normal_C の通信を遮断する．各ノードの遮断時の limit 値を表 5.6 に示す．

表 5.5: q = 70 でのシナリオ B の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	8,561	0
ゲートウェイから遮断された通信	856	417

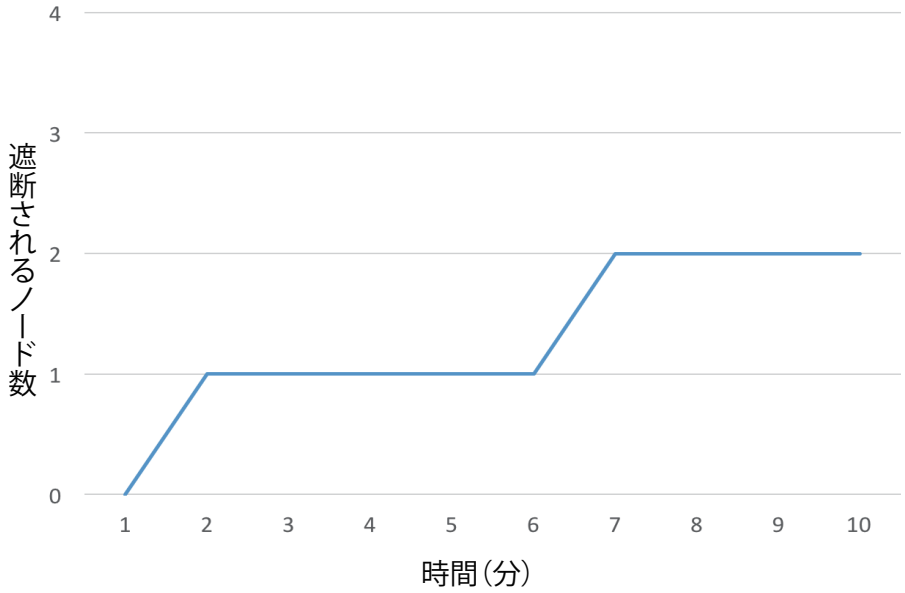

図 5.3: q = 70 でのシナリオ B の分単位での遮断されたノード数の推移

表 5.6: q = 70 でのシナリオ B の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_B	80	108
Normal_C	184.26.229.110	6

5.4.2　q = 75 の場合

シナリオ A - Linux で DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 7,347 パケットである．各ノードは正常な通信を 1,347 パケット送り，Attacker_A は不正な通信を 6,000 パケット送信した．その結果，本システムが 6,316 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.7 のようになった．誤検知率は 23%となり，不検知率は 0%となる．遮断されるノード数の推移を図 5.4 に示す．実験を開始して 1 分 59 秒後に Attacker_A の通信を遮断し，2 分 41 秒後に Normal_C の通信を遮断し，7 分 49 秒後に Normal_D の通信を遮断する．各ノードの遮断時の limit 値を表 5.8 に示す．

表 5.7: q = 75 でのシナリオ A の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	1,031	0
ゲートウェイから遮断された通信	316	6,000

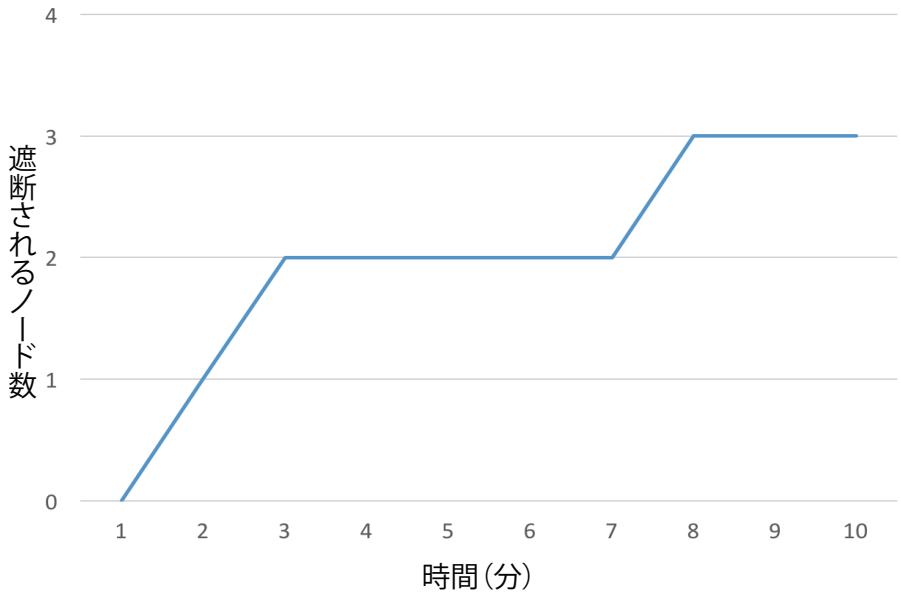

図 5.4: q = 75 でのシナリオ A の分単位での遮断されたノード数の推移

シナリオ B - DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 13,332 パケットである．Attacker_B を除く各ノードは正常な通信を 9,752 パケット送り，Attacker_B は不正な通信を 3,580 パケット送信した．その結果，本システムが 4,449 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.9 のようになった．誤検知率は 25%となり，不検知率は 45%となる．遮断されるノード数の推移を図 5.5 に示す．実験を開始して 1 分 59 秒後に Normal_D の通信を遮断する．5 分 18 秒後に Attacker_B の通信を遮断する．5 分 58 秒後に Normal_C を

表 5.8: q = 75 でのシナリオ A の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_A	10.1.3.80	1
Normal_C	216.58.196.234	1
Normal_D	10.1.3.80	1

遮断する．各ノードの遮断時の limit 値を表 5.10 に示す．

表 5.9: q = 75 でのシナリオ B の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	7,270	1,613
ゲートウェイから遮断された通信	2,482	1,967

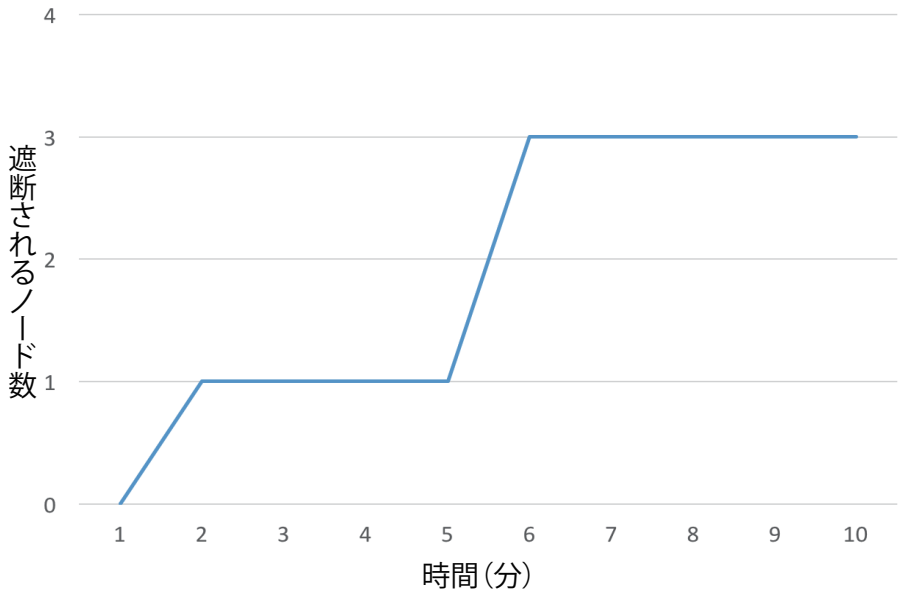

図 5.5: q = 75 でのシナリオ B の分単位での遮断されたノード数の推移

表 5.10: q = 75 でのシナリオ B の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_B	10.1.167.31	1
Normal_C	133.237.48.32	1
Normal_D	10.1.3.80	1

5.4.3　q = 80 の場合

シナリオ A - Linux で DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 10,462 パケットである．各ノードは正常な通信を 4,462 パケット送り，Attacker_A は不正な通信を 6,000 パケット送信した．その結果，本システムが 6,863 パケットの通信を遮断した．10 分間の通信の分類結

果は表 5.11 のようになった．誤検知率は 19%となり，不検知率は 0%となる．遮断されるノード数の推移を図 5.6 に示す．実験を開始して 2 分 3 秒後に Attacker_A の通信を遮断し，6 分 34 秒後に Normal_C の通信を遮断し，6 分 57 秒後に Normal_D の通信を遮断．各ノードの遮断時の limit 値を表 5.12 に示す．

表 5.11: q = 80 でのシナリオ A の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	3,599	0
ゲートウェイから遮断された通信	863	6,000

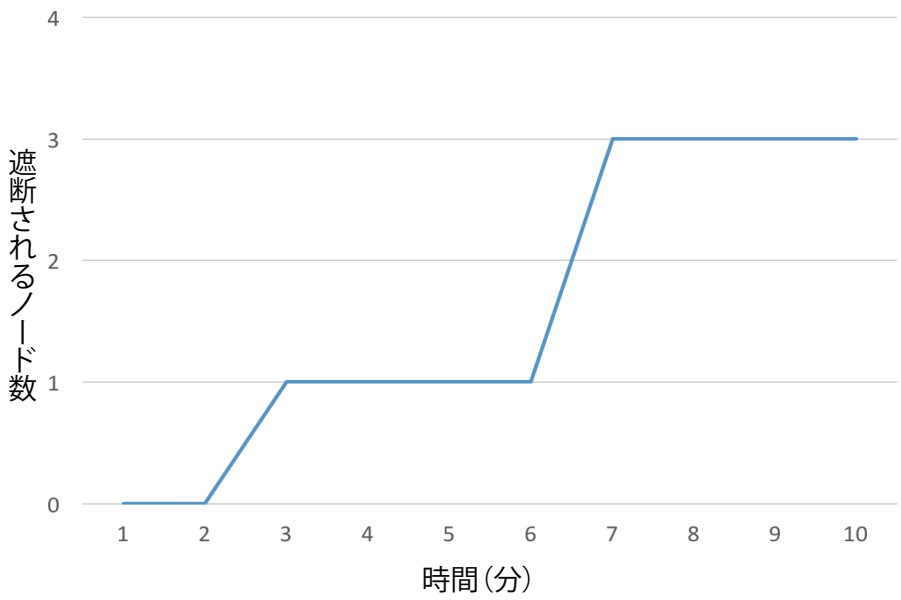

図 5.6: q = 80 でのシナリオ A の分単位での遮断されたノード数の推移

表 5.12: q = 80 でのシナリオ A の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_A	10.1.3.80	1
Normal_C	104.244.42.67	1
Normal_D	10.1.3.80	1

シナリオ B - DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 9,518 パケットである．Attacker_B を除く各ノードは正常な通信を 2,129 パケット送り，Attacker_B は不正な通信を 27,389 パケット送信した．その結果，本システムが 2,245 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.13 のようになった．誤検知率は 24%となり，不検知率は 76%となる．遮断されるノード数の推移を図 5.7 に示す．実験を開始して 13 秒後に Normal_D の通信を遮断する．39 秒後に Normal_C の通信を遮断する．5 分 07 秒後に Attacker_B を遮断する．各ノードの遮断時の limit 値を表 5.14 に示す．

表 5.13: q = 80 でのシナリオ B の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	1,621	5,652
ゲートウェイから遮断された通信	508	1,737

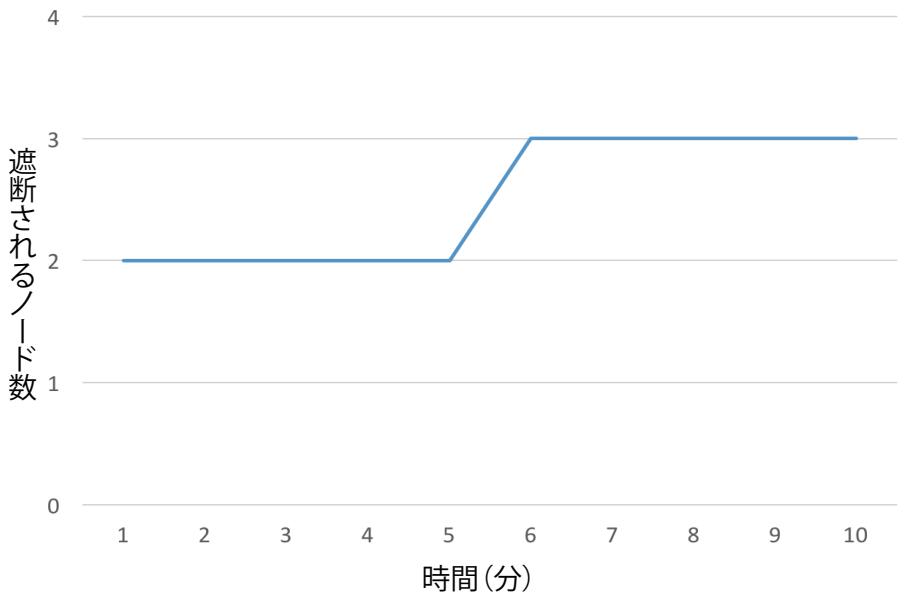

図 5.7: q = 80 でのシナリオ B の分単位での遮断されたノード数の推移

表 5.14: q = 80 でのシナリオ B の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_B	10.1.167.31	1
Normal_C	216.58.196.234	1
Normal_D	10.1.3.80	1

5.4.4　q = 90 の場合

シナリオ A - Linux で DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 12,301 パケットである．各ノードは正常な通信を 6,301 パケット送り，Attacker_A は不正な通信を 6,000 パケット送信した．その結果，本システムが 7,550 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.15 のようになった．誤検知率は 25%となり，不検知率は 0%となる．遮断されるノード数の推移を図 5.8 に示す．実験を開始して 1 分 59 秒後に Normal_D の通信を遮断し，4 分 22 秒後に Attacker_A の通信を遮断し，5 分 45 秒後に Normal_C の通信を遮断する．各ノードの遮断時の limit 値を表 5.16 に示す.

表 5.15: q = 90 でのシナリオ A の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	3,201	0
ゲートウェイから遮断された通信	1,550	6,000

表 5.16: q = 90 でのシナリオ A の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_A	10.1.3.80	1
Normal_C	172.217.26.98	6
Normal_D	10.1.3.80	1

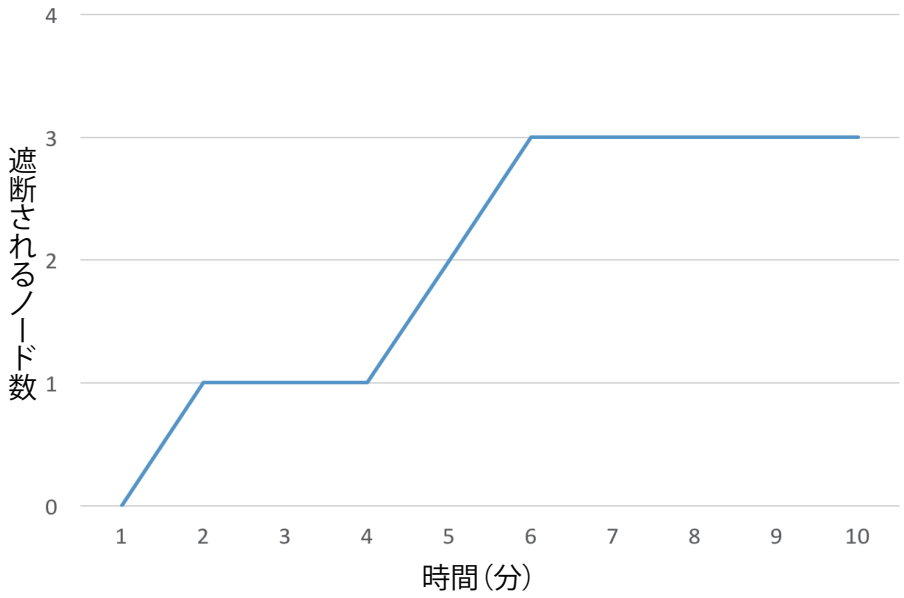

図 5.8: q = 90 でのシナリオ A の分単位での遮断されたノード数の推移

シナリオ B - DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 9,300 パケットである．Attacker_B を除く各ノードは正常な通信を 1,159 パケット送り，Attacker_B は不正な通信を 8,141 パケット送信した．その結果，本システムが 1,893 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.17 のようになった．誤検知率は 17%となり，不検知率は 79%となる．遮断されるノード数の推移を図 5.9 に示す．実験を開始して 5 分 23 秒後に Attacker_B の通信を遮断する．5 分 50 秒後に Normal_C の通信を遮断する．8 分 21 秒後に Normal_D を遮断する．各ノードの遮断時の limit 値を表 5.18 に示す．

表 5.17: q = 90 でのシナリオ B の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	957	6,450
ゲートウェイから遮断された通信	202	1,691

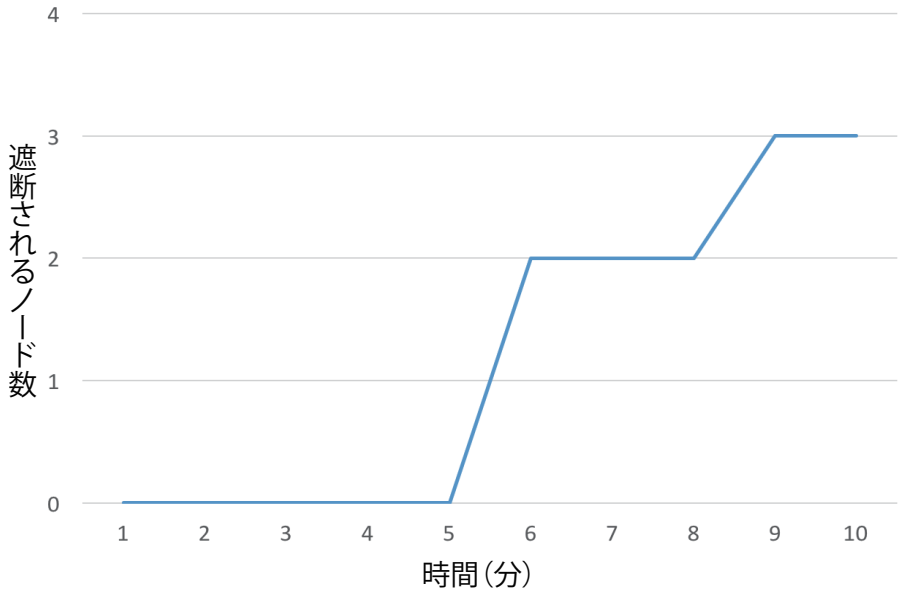

図 5.9: q = 90 でのシナリオ B の分単位での遮断されたノード数の推移

表 5.18: q = 90 でのシナリオ B の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_B	10.1.167.31	1
Normal_C	216.58.199.234	1
Normal_D	10.1.3.80	1

5.4.5　q = 100 の場合

シナリオ A - Linux で DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 6,762 パケットである．各ノードは正常な通信を 562 パケット送り，Attacker_A は不正な通信を 6,000 パケット送信した．その結果，本システムが 6,200 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.19 のようになった．誤検知率は 5%となり，不検知率は 0%となる．遮断されるノード数の推移を図 5.10 に示す．実験を開始して 1 分 33 秒後に Normal_D の通信を遮断し，5 分 49 秒後に Normal_C の通信を遮断し，6 分 32 秒後に Normal_C を遮断する．各ノードの遮断時の limit 値を表 5.20 に示す．

表 5.19: q = 100 でのシナリオ A の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	362	0
ゲートウェイから遮断された通信	200	6,000

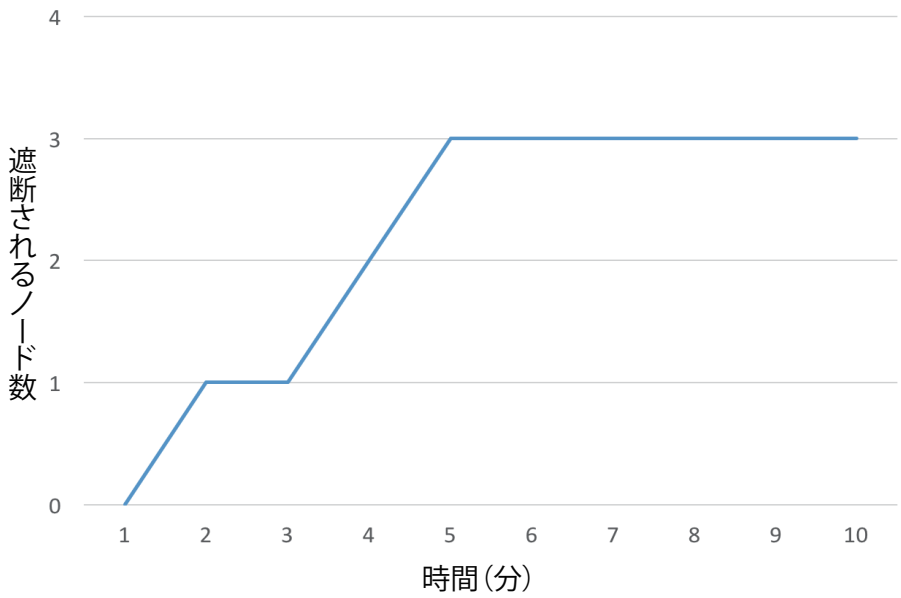

図 5.10: q = 100 でのシナリオ A の分単位での遮断されたノード数の推移

シナリオ B - DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 7,512 パケットである．Attacker_B を除く各ノードは正常な通信を 3,704 パケット送り，Attacker_B は不正な通信を 3,808 パケット送信した．その結果，本システムが 1,766 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.21 のようになった．誤検知率は 5%となり，不検知率は 42%となる．遮断されるノード数の推移を図 5.11 に示す．実験を開始して 1 分 33 秒後に Normal_D の通信を遮断する．5 分 49 秒後に Attacker_B の通信を遮断する．6 分 32 秒後に Normal_C

表 5.20: q = 100 でのシナリオ A の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_A	10.1.3.80	1
Normal_C	216.58.199.234	1
Normal_D	10.1.3.80	1

を遮断する．各ノードの遮断時の limit 値を表 5.22 に示す．

表 5.21: q = 100 でのシナリオ B の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	3,522	2,224
ゲートウェイから遮断された通信	182	1,584

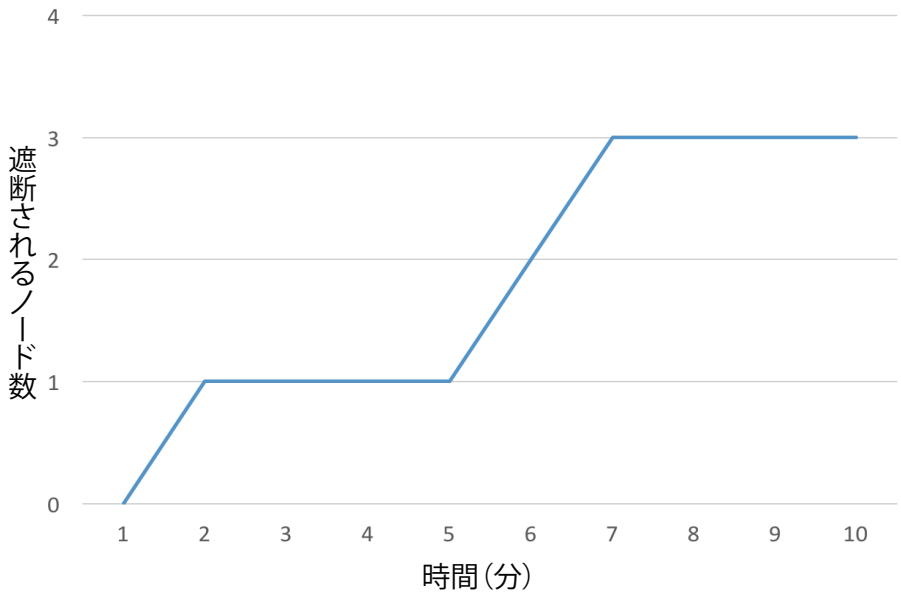

図 5.11: q = 100 でのシナリオ B の分単位での遮断されたノード数の推移

表 5.22: q = 100 でのシナリオ B の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_B	10.1.167.31	1
Normal_C	216.58.197.3	2
Normal_D	10.1.3.80	1

5.4.6　実験 A の考察

攻撃を完全に遮断できたのは $q = 70$ のときであったが，誤検知率も高い．全ての実験で攻撃を遮断することに成功したが，一般ユーザのノードも通信を遮断してしまうことになった．また，シナリオ A で Attacker_A が攻撃を開始する前に，一般ユーザが利用している際に遮断が行われることが多く，シナリオ A の不検知率を下げる一因となった．

5.5　実験 B

実験 A を通して，$q = 70$ の閾値がもっとも攻撃ノードの遮断に成功していたが，誤検知率も高かった．そこで，$q = 70$ の値から $q = 75$ の値について再度実験を行い，最適な q が無いか調査する．

5.5.1　q = 70 の場合

シナリオ A - Linux で DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 18,113 パケットである．各ノードは正常な通信を 12,005 パケット送り，Attacker_A は不正な通信を 6,108 パケット送信した．その結果，本システムが 7,162 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.23 のようになった．誤検知率は 10%となり，不検知率は 3%となる．遮断されるノード数の推移を図 5.12 に示す．実験を開始して 1 分 27 秒後に Normal_D の通信を遮断し，5 分 4 秒後に Attacker_A の通信を遮断し，6 分 8 秒後に Normal_C を遮断する．各ノードの遮断時の limit 値を表 5.24 に示す．

表 5.23: q = 70 でのシナリオ A の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	10,789	162
ゲートウェイから遮断された通信	1,216	5,946

表 5.24: q = 70 でのシナリオ A の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_A	10.1.167.31	1
Normal_C	184.26.239.83	1
Normal_D	10.1.3.80	1

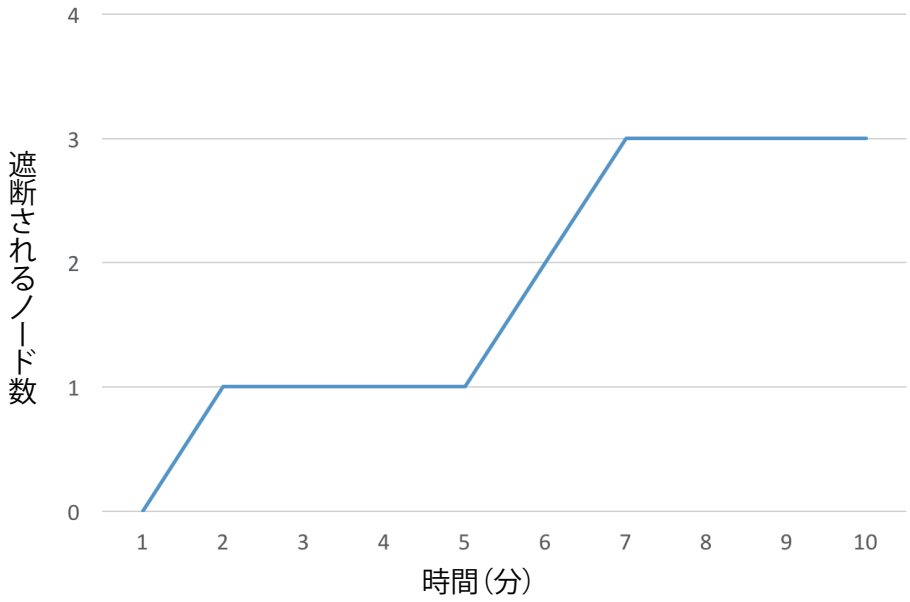

図 5.12: q = 70 でのシナリオ A の分単位での遮断されたノード数の推移

シナリオ B - DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 5,792 パケットである．Attacker_B を除く各ノードは正常な通信を 3,931 パケット送り，Attacker_B は不正な通信を 1,861 パケット送信した．その結果，本システムが 2,591 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.25 のようになった．誤検知率は 37%となり，不検知率は 39%となる．遮断されるノード数の推移を図 5.13 に示す．実験を開始して 1 分 19 秒後に Normal_D の通信を遮断する．5 分 3 秒後に Attacker_B の通信を遮断する．5 分 24 秒後に Normal_C を遮断する．各ノードの遮断時の limit 値を表 5.26 に示す．

表 5.25: q = 70 でのシナリオ B の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	2,484	717
ゲートウェイから遮断された通信	1,447	1,144

5.5.2　q = 71 の場合

シナリオ A - Linux で DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 14,369 パケットである．各ノードは正常な通信を 8,369 パケット送り，Attacker_A は不正な通信を 6,000 パケット送

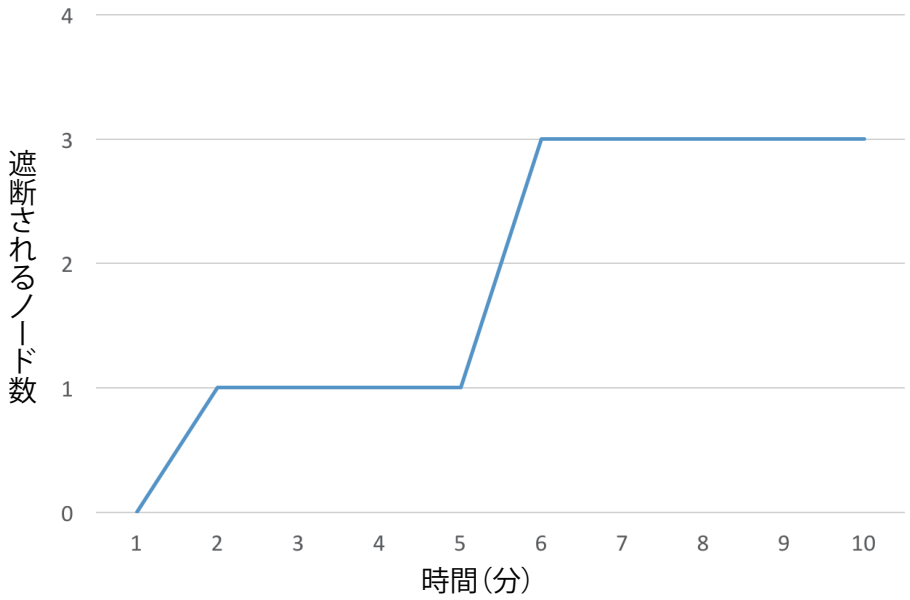

図 5.13: q = 70 でのシナリオ B の分単位での遮断されたノード数の推移

表 5.26: q = 70 でのシナリオ B の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_B	10.1.167.31	1
Normal_C	184.26.229.110	6
Normal_D	10.1.3.80	1

信した．その結果，本システムが 6,000 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.27 のようになった．誤検知率は 20%となり，不検知率は 0%となる．遮断されるノード数の推移を図 5.14 に示す．実験を開始して 46 秒後に Normal_D の通信を遮断し，4 分 49 秒後に Attacker_A の通信を遮断し，5 分 51 秒後に Normal_C を遮断する．各ノードの遮断時の limit 値を表 5.28 に示す．

表 5.27: q = 71 でのシナリオ A の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	6,675	0
ゲートウェイから遮断された通信	1,694	6,000

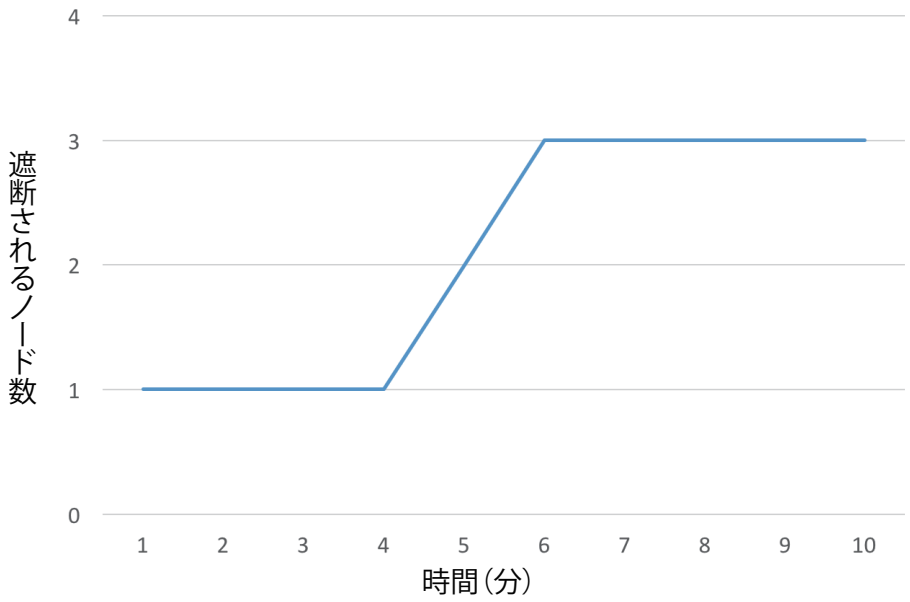

図 5.14: q = 71 でのシナリオ A の分単位での遮断されたノード数の推移

表 5.28: q = 71 でのシナリオ A の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_A	10.1.3.80	1
Normal_C	52.192.132.119	1
Normal_D	10.1.3.80	1

シナリオ B - DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 2,512 パケットである．Attacker_B を除く各ノードは正常な通信を 1,816 パケット送り，Attacker_B は不正な通信を 696 パケット送信した．その結果，本システムが 1,414 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.29 のようになった．誤検知率は 43%となり，不検知率は 10%となる．遮断されるノード数の推移を図 5.15 に示す．実験を開始して 1 分 20 秒後に Normal_D の通信を遮断する．2 分 24 秒後に Normal_D の通信を遮断する．4 分 49 秒後に Attacker_B を遮断する．各ノードの遮断時の limit 値を表 5.30 に示す．

表 5.29: q = 71 でのシナリオ B の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	3,522	2,224
ゲートウェイから遮断された通信	182	1,584

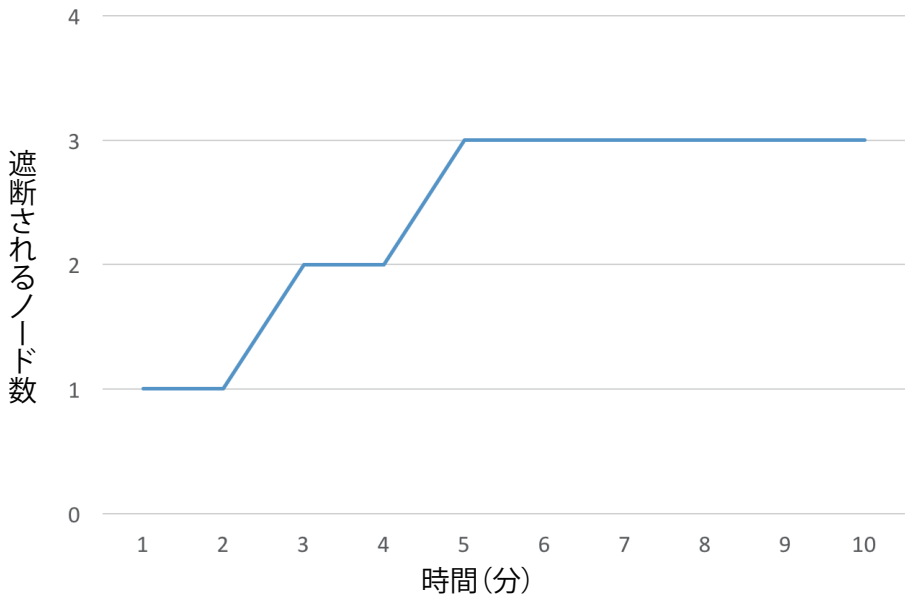

図 5.15: q = 100 でのシナリオ B の分単位での遮断されたノード数の推移

表 5.30: q = 71 でのシナリオ B の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_B	10.1.167.31	1
Normal_C	216.58.197.3	2
Normal_D	10.1.3.80	1

5.5.3　q = 72 の場合

シナリオ A - Linux で DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 10,765 パケットである．各ノードは正常な通信を 4,765 パケット送り，Attacker_A は不正な通信を 6,000 パケット送信した．その結果，本システムが 7,452 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.31 のようになった．誤検知率は 30%となり，不検知率は 0%となる．遮断されるノード数の推移を図 5.16 に示す．実験を開始して 50 秒後に Attacker_A の通信を遮断し，1 分 27 秒後に Normal_D の通信を遮断し，5 分 17 秒後に Normal_D を遮断する．各ノードの遮断時の limit 値を表 5.32 に示す．

表 5.31: q = 72 でのシナリオ A の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	3,313	0
ゲートウェイから遮断された通信	1,452	6,000

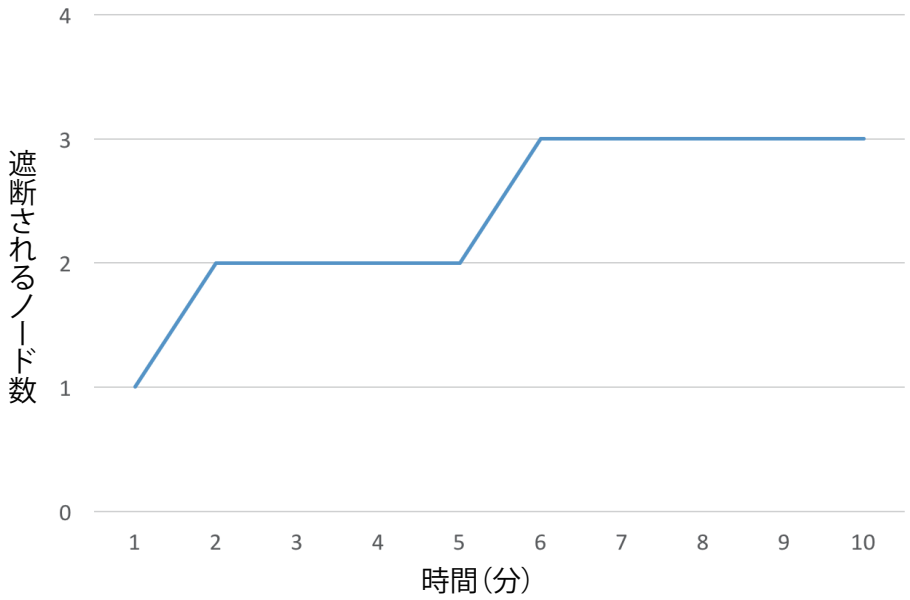

図 5.16: q = 72 でのシナリオ A の分単位での遮断されたノード数の推移

シナリオ B - DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 8,543 パケットである．Attacker_B を除く各ノードは正常な通信を 6,443 パケット送り，Attacker_B は不正な通信を 2,100 パケット送信した．その結果，本システムが 2,337 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.33 のようになった．誤検知率は 17%となり，不検知率は 41%となる．遮断されるノード数の推移を図 5.17 に示す．実験を開始して 56 秒後に Normal_D の通信を遮断する．5 分 4 秒後に Attacker_B の通信を遮断する．5 分 34 秒後に Normal_C を遮

表 5.32: q = 72 でのシナリオ A の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_A	10.1.3.80	1
Normal_C	133.237.16.180	1
Normal_D	10.1.3.80	1

断する．各ノードの遮断時の limit 値を表 5.34 に示す．

表 5.33: q = 72 でのシナリオ B の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	5,336	870
ゲートウェイから遮断された通信	1,107	1,230

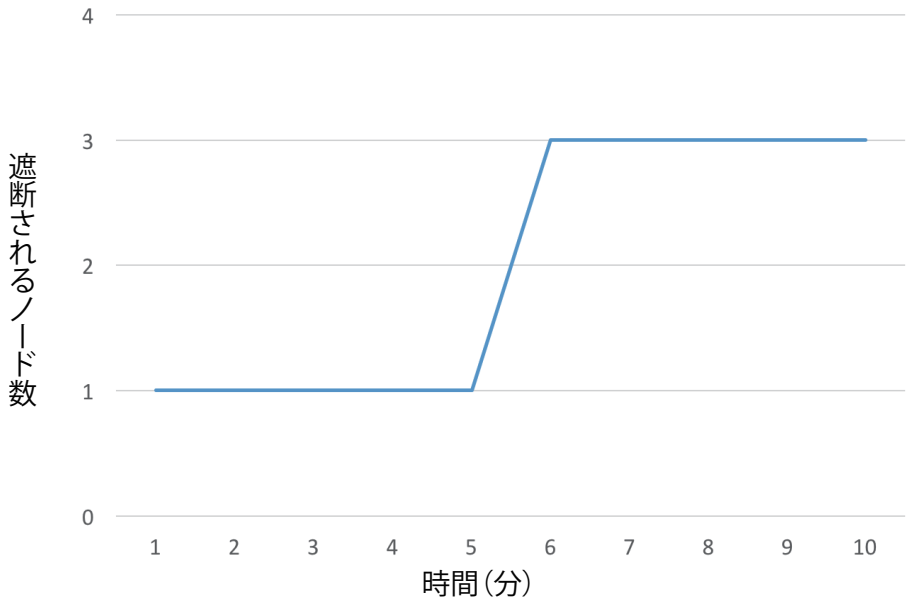

図 5.17: q = 72 でのシナリオ B の分単位での遮断されたノード数の推移

表 5.34: q = 72 でのシナリオ B の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_B	10.1.167.31	1
Normal_C	133.237.17.2	1
Normal_D	10.1.3.80	1

5.5.4　q = 73 の場合

シナリオ A - Linux で DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 12,175 パケットである．各ノードは正常な通信を 6,175 パケット送り，Attacker_A は不正な通信を 6,000 パケット送信した．その結果，本システムが 6,873 パケットの通信を遮断した．10 分間の通信の分類結

果は表 5.35 のようになった．誤検知率は 14%となり，不検知率は 0%となる．遮断されるノード数の推移を図 5.18 に示す．実験を開始して 2 分 56 秒後に Normal_D の通信を遮断し，4 分 46 秒後に Attacker_A の通信を遮断し，5 分 41 秒後に Normal_C を遮断する．各ノードの遮断時の limit 値を表 5.36 に示す．

表 5.35: q = 73 でのシナリオ A の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	5,302	0
ゲートウェイから遮断された通信	873	6,000

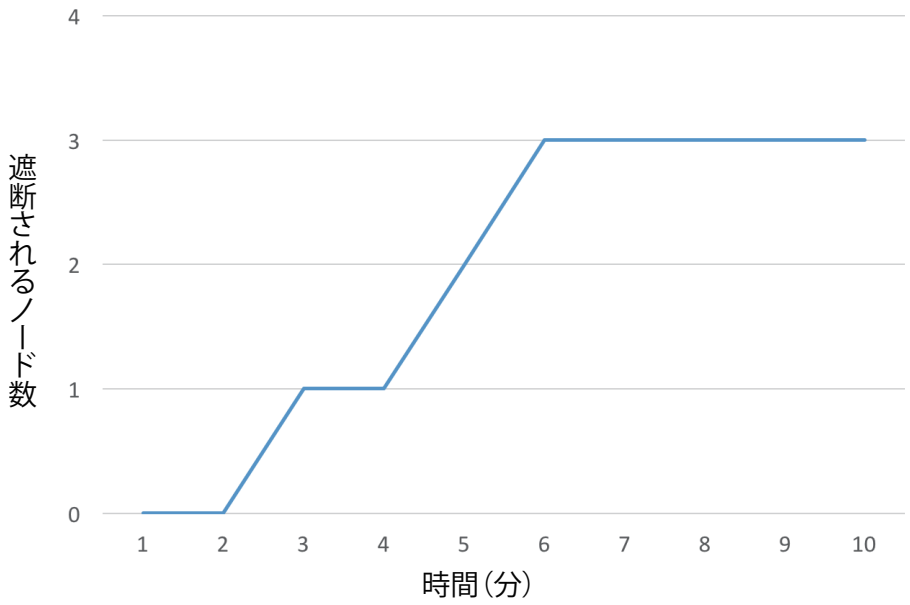

図 5.18: q = 73 でのシナリオ A の分単位での遮断されたノード数の推移

表 5.36: q = 73 でのシナリオ A の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_A	10.1.3.80	1
Normal_C	184.26.235.48	6
Normal_D	172.16.234.1	1

シナリオ B - DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 1,928 パケットである．Attacker_B を除く各ノードは正常な通信を 614 パケット送り，Attacker_B は不正な通信を 1,314 パケット送信した．その結果，本システムが 1,404 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.37 のようになった．誤検知率 34%となり，不検知率は 24%となる．遮断されるノード数の推移を図 5.19 に示す．実験を開始して 8 秒後に Normal_C の通信を遮断する．51 秒後に Normal_D の通信を遮断する．5 分 00 秒後に Attacker_B を遮断する．各ノードの遮断時の limit 値を表 5.38 に示す．

表 5.37: q = 73 でのシナリオ B の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	210	314
ゲートウェイから遮断された通信	404	1,000

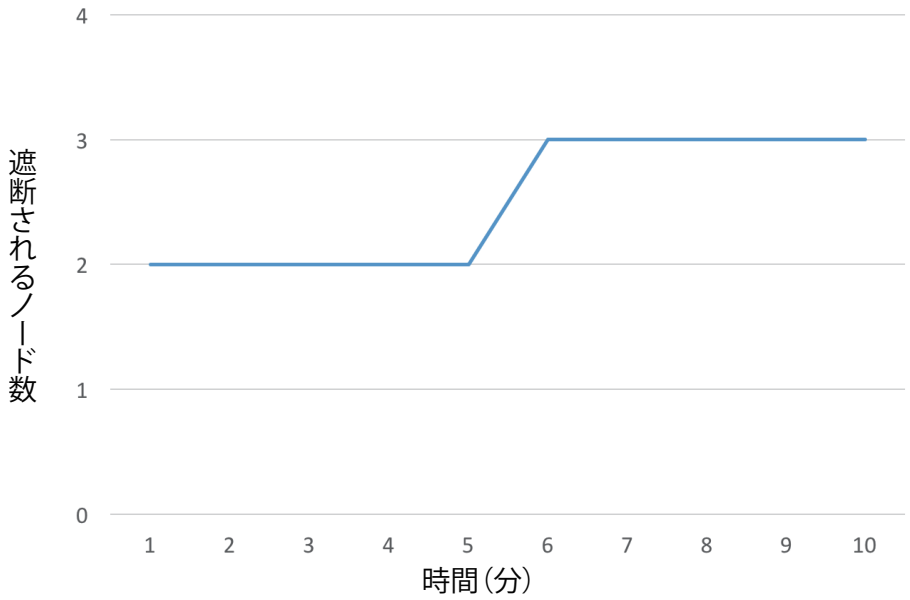

図 5.19: q = 73 でのシナリオ B の分単位での遮断されたノード数の推移

表 5.38: q = 73 でのシナリオ B の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_B	10.1.167.31	1
Normal_C	104.244.42.69	1
Normal_D	10.1.3.80	1

5.5.5　q = 74 の場合

シナリオ A - Linux で DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 7,694 パケットである．各ノードは正常な通信を 2,694 パケット送り，Attacker_A は不正な通信を 6,000 パケット送信した．その結果，本システムが 6,432 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.39 のようになった．誤検知率は 47%となり，不検知率は 0%となる．遮断されるノード数の推移を図 5.20 に示す．実験を開始して 1 分 14 秒後に Attacker_A の通信を遮断し，2 分 9 秒後に Normal_D の通信を遮断し，2 分 56 秒後に Normal_C を遮断する．各ノードの遮断時の limit 値を表 5.40 に示す．

表 5.39: q = 74 でのシナリオ A の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	1,262	0
ゲートウェイから遮断された通信	1,432	6,000

表 5.40: q = 74 でのシナリオ A の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_A	10.1.3.80	1
Normal_C	172.217.26.106	1
Normal_D	10.1.3.80	1

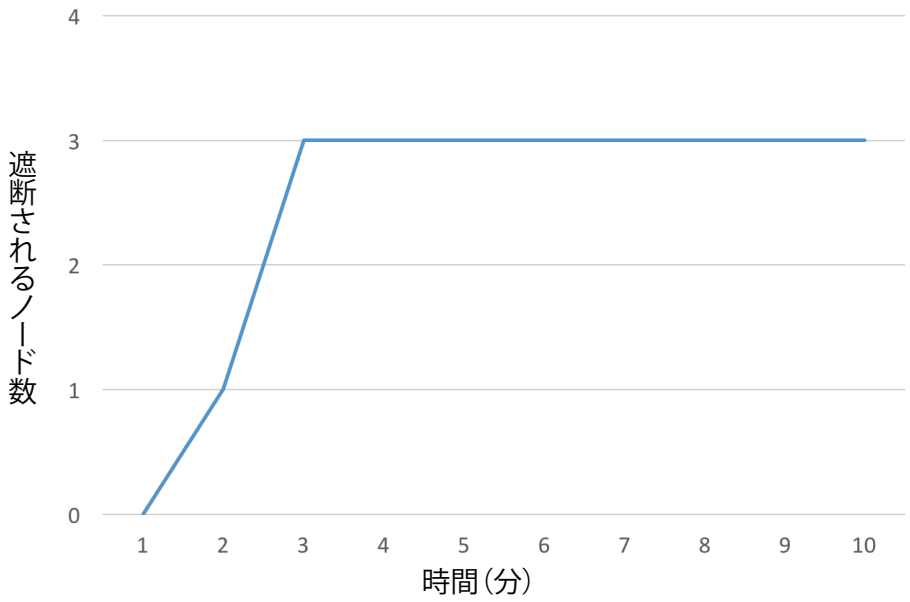

図 5.20: q = 74 でのシナリオ A の分単位での遮断されたノード数の推移

シナリオ B - DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 5,676 パケットである．Attacker_B を除く各ノードは正常な通信を 3,898 パケット送り，Attacker_B は不正な通信を 1,778 パケット送信した．その結果，本システムが 2,588 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.41 のようになった．誤検知率は 38%となり，不検知率は 37%となる．遮断されるノード数の推移を図 5.21 に示す．実験を開始して 1 分 12 秒後に Normal_D の通信を遮断する．5 分 2 秒後に Attacker_B の通信を遮断する．5 分 24 秒後に Normal_C を遮断する．各ノードの遮断時の limit 値を表 5.42 に示す．

表 5.41: q = 74 でのシナリオ B の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	2,426	662
ゲートウェイから遮断された通信	1,472	1,116

5.5.6　q = 75 の場合

シナリオ A - Linux で DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 7,321 パケットである．各ノードは正常な通信を 1321 パケット送り，Attacker_A は不正な通信を 6,000 パケット送信

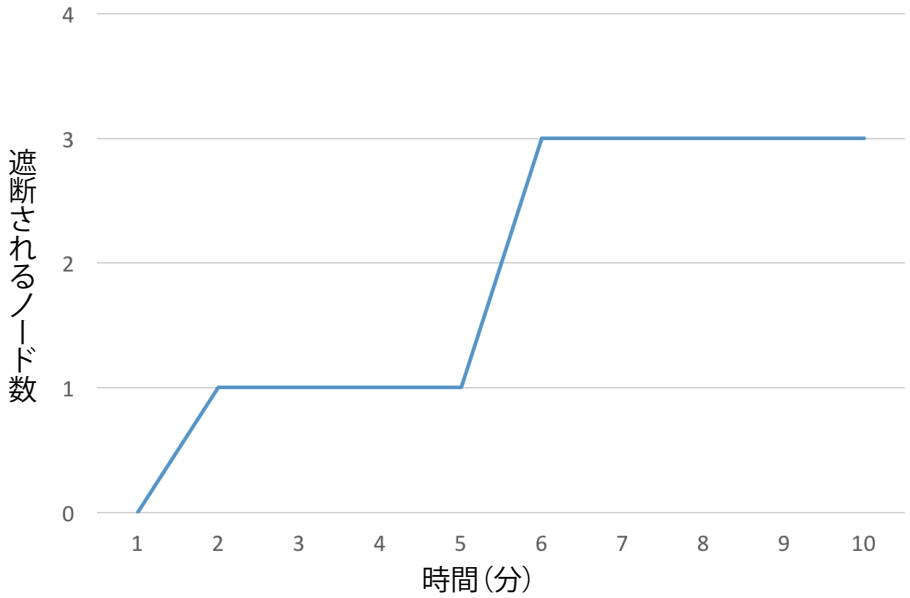

図 5.21: q = 74 でのシナリオ B の分単位での遮断されたノード数の推移

表 5.42: q = 74 でのシナリオ B の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_B	10.1.167.31	1
Normal_C	133.237.16.65	1
Normal_D	10.1.3.80	1

した．その結果，本システムが 6,536 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.43 のようになった．誤検知率は 41%となり，不検知率は 0%となる．遮断されるノード数の推移を図 5.22 に示す．実験を開始して 33 秒後に Normal_C の通信を遮断し，1 分 44 秒後に Normal_D の通信を遮断し，4 分 14 秒後に Attacker_A を遮断する．各ノードの遮断時の limit 値を表 5.44 に示す．

表 5.43: q = 75 でのシナリオ A の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	785	0
ゲートウェイから遮断された通信	536	6,000

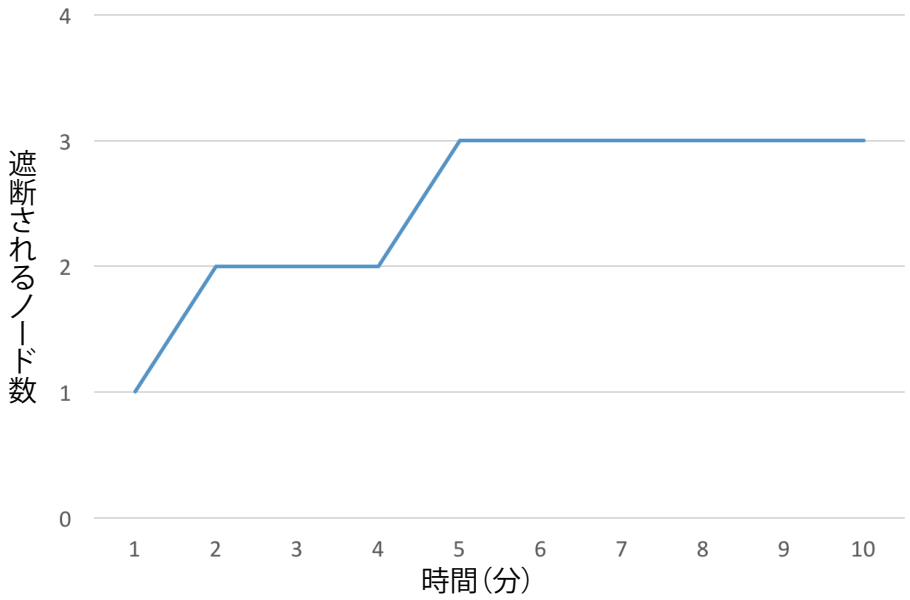

図 5.22: q = 75 でのシナリオ A の分単位での遮断されたノード数の推移

表 5.44: q = 75 でのシナリオ A の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_A	10.1.3.80	1
Normal_C	216.58.196.234	1
Normal_D	10.1.3.80	1

シナリオ B - DoS 攻撃ツールが利用されている場合

悪意あるユーザは実験開始の 5 分後に攻撃を始める．総通信量は 4,253 パケットである．Attacker_B を除く各ノードは正常な通信を 1,673 パケット送り，Attacker_B は不正な通信を 2,580 パケット送信した．その結果，本システムが 1,696 パケットの通信を遮断した．10 分間の通信の分類結果は表 5.45 のようになった．誤検知率は 21%となり，不検知率は 52%となる．遮断されるノード数の推移を図 5.23 に示す．実験を開始して 39 秒後に Normal_D の通信を遮断する．3 分 3 秒後に Normal_C の通信を遮断する．5 分 5 秒後に Attacker_B を遮断する．各ノードの遮断時の limit 値を表 5.46 に示す．

表 5.45: q = 75 でのシナリオ B の実験結果の通信分類

	正常な通信	不正な通信
ゲートウェイを通過した通信	1,324	1,233
ゲートウェイから遮断された通信	349	1,347

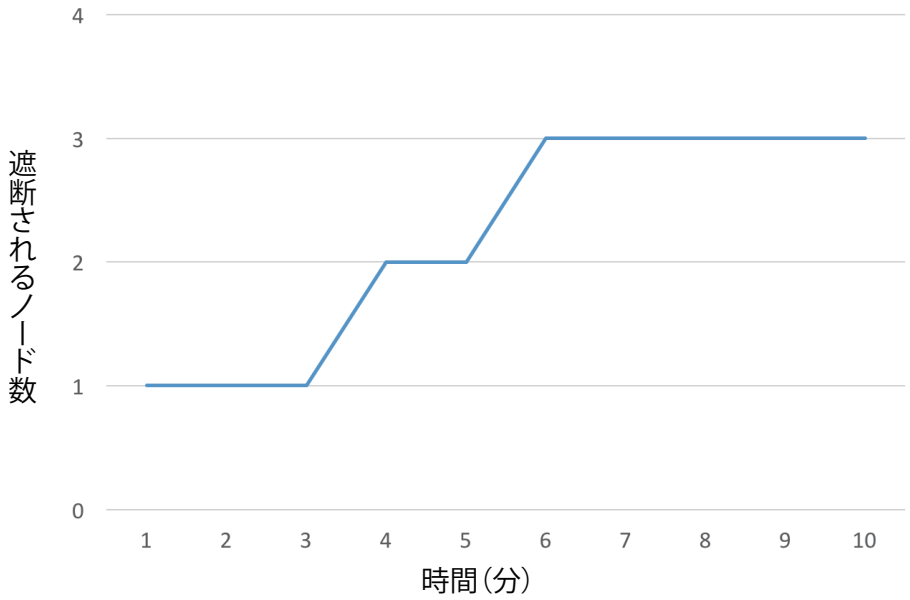

図 5.23: q = 75 でのシナリオ B の分単位での遮断されたノード数の推移

表 5.46: q = 75 でのシナリオ B の各ノードの遮断時の limit 値

ノード名	宛先 IP アドレスまたは宛先ポート番号	limit 値
Attacker_B	10.1.167.31	1
Normal_C	104.244.42.1	1
Normal_D	10.1.3.80	1

5.5.7　実験 B の考察

全ての q パーセント点で，シナリオ A での不検知パケットが無かった．理由として，Attacker_A が一般ユーザに利用されている際に受ける制限が厳しすぎることが挙げられる．シナリオ A の誤検知率と，シナリオ B の不検知率を見て，$q = 73$ が最も最適な点であると言える．また，q の値が上がるごとに閾値は緩くなるはずだが，誤検知率は下がることは無かった．

5.6　まとめ

実験 AB を通して，$q = 73$ を最も最適な q として挙げることができる．また，全実験を通して宛先 IP アドレスを対象としたリアルタイムフィルタリングによってノードが遮断された．limit 値を算出する際に閾値を 1 時間の分数である 60 で割っているが，割る数が大きいため殆どの送信先 IP アドレスを対象とした limit 値が 1 になってしまい，判定が厳しくなりすぎたことが原因として挙げられる．limit 値が低すぎた場合の対策を考える必要がある．シナリオ B の不検知率が高い原因として，デフォルトゲートウェイの負荷増加による遮断ルール適

用プログラムの低速化が挙げられる．また，閾値設定が宛先 IP アドレスによっては低すぎることもあり，正常な量のパケットでも閾値を突破してしまうことで，誤検知率を上げてしまうことになった．攻撃ツールのパケット速度と正常なユーザのパケット速度を正確に求める必要がある．

第6章　おわりに

本研究では，DoS 攻撃の攻撃者になり得るホストの正常な通信データのログを蓄積し，そこから閾値を求め，通信の流量制限と悪性ホストのアクセス遮断を行うフィルタリングルールを設定するゲートウェイを構築した．構築したゲートウェイで実際にホストが攻撃者に利用された場合のシミュレーションを行い，悪性ホストを遮断することができることを確認した．

本研究では送信元 MAC アドレスを用いてホストの制限を行った．しかし，送信元 MAC アドレスを偽装することが可能であることから，悪意あるユーザが自分のホストを他のホストの MAC アドレスに偽装する可能性がある．ARP Spoofing 等の MAC アドレスを偽装する手段を防止してから，本システムを利用する必要がある．

また，ユーザの正常な通信にもフィルタリングをしてしまうことがあることや，本システム自体に負荷がかかることによりゲートウェイとして機能しなくなる可能性も考慮しなくてはならない．フィルタリングの状況によっては，フィルタが緩すぎることもあり，その際に不正な通信をフィルタできないことを，今後の課題として挙げる．

謝辞

本研究を進めるにあたり，多数の助言と手厚いご指導をいただきました関西大学総合情報学部の小林孝史准教授に深く御礼申し上げます．今後ともよろしくお願い致します．また，研究外の基礎的な事柄からご指導を頂きました小林研究室の先輩方に感謝申し上げます．同研究室の同期・後輩の皆様には，研究を進める際に様々な励ましを頂きました．感謝申し上げます．

参考文献，参考URL等

[1] Akamai Technologies, ”akamai ’ s [state of the internet] / security Q3 2016 report”, https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q3-2016-state-of-the-internet-security-report.pdf, 2016年11月30日確認.

[2] IPA, ”2015 年度情報セキュリティの脅威に対する意識調査”, http://www.ipa.go.jp/files/000050002.pdf, 2016年11月30日確認.

[3] ITmedia エンタープライズ，セキュリティニュースサイトに史上最大規模の DDoS 攻撃、1Tbps のトラフィックも，http://www.itmedia.co.jp/enterprise/articles/1609/26/news047.html, 2017 年 1 月22日確認.

[4] 池渕遼馬，“Webアクセスログに基づくファイアウォールの動的な制御に関する研究”，平成25年関西大学卒業論文.

[5] Red Hat Customer Portal, Why do we see long MAC address in iptables log message?, https://access.redhat.com/solutions/70465, 2017年1月24日確認.

[6] Ubuntu Manpage, tgn - a network traffic generator, http://manpages.ubuntu.com/manpages/trusty/man1/tgn.1.html, 2017 年 2 月 6日確認.

[7] NewEraCracker, LOIC, https://github.com/NewEraCracker/LOIC, 2017年2月6日確認.