NANZAN-TR-2009-04

ネットワークセキュリティのためのトラフィック制限システム

福井麻美　　後藤邦夫

November 2009

Technical Report of the Nanzan Academic Society
Information Sciences and Engineering

ネットワークセキュリティのためのトラフィック制限システム

福井 麻美　　後藤邦夫
南山大学 大学院 数理情報研究科

和文概要　　インターネットから LAN に配置されたサーバ等への通信には，正常な要求，悪意に基づく攻撃，の他に，判定が難しいものがある．判定が難しい通信に対し，通過と遮断の中間的な通信制限が必要となる．そこで，本研究では，ネットワークトラフィックを制限するシステム (GateKeeper と呼ぶ) を提案した．制限は，数秒以上の遅延，利用バンド幅，ランダムパケットロス，遮断とした．GateKeeper は，Linux のユーザ空間プロセスで動作するブリッジプログラムとして実現し，DoS などの攻撃に対する有効性を擬似攻撃実験で評価した．その結果，あやしい通信を抑制するためにこれらの制限が有効であることがわかった．攻撃は，本システムの外部に配置した侵入検知システムまたは Web，メイルなどのアプリケーションサーバで検知し，検知したホストからこの GateKeeper に制限指示があることを前提とする．

キーワード: セキュリティ，インターネット，トラフィック制限

1. はじめに

侵入検知システム (IDS) の利用が普及してきたが，多数の IDS からの警告 (アラート) から本当の攻撃を見つけることは困難である．管理者が攻撃とみなした場合，ファイアウォールにおいて，該当する通信を遮断することが一般にとられる対策であるが，もし，攻撃に思えるが実は正常な通信を遮断すると通信障害となる．正常な通信か攻撃の判定が難しい場合，まずその通信を抑制することがフェイルセーフになる．いくつかの商用の侵入検知と予防システム [4] は，トラフィック制限機能を備えている．しかし，検知と予防がひとつの箱にふくめられていて通信制限の効果をくわしく調べたり，色々な攻撃に対する制限ルールの設定ポリシを開発することは困難である．

そこで本研究では，特に DoS タイプの攻撃に対するトラフィック制限の有効性を調べるために，ソフトウェアでネットワークトラフィック制限システム (GateKeeper と呼ぶ) を実現する．GateKeeper は，Linux のユーザ空間でブリッジとして動作し，外部に設置された IDS やサーバからの指示で設定された制限フィルタルールにしたがって，入力フレームを分類し，素通し，破棄，制限付き中継のためにカスタムフレームキューに入れる．フレームキューは，遅延，帯域制限，ランダムフレーム損失を表現する．本研究で使用するフレームキューとタイマの実現方法は，著者らの開発中のネットワークエミュレータ [3] に基づく．

GateKeeper のブリッジとしての転送性能とトラフィック制限の効果は，数台の PC を用いた実験で評価する．本システムは制限ルールの開発または簡便な侵入予防システムの一部として有用である．

第 2 節では，提案するシステムの概要，第 3 節で実現方法の詳細，第 4 節で性能と擬似攻撃実験の結果を述べる．

2. システムの概要

本節では，提案するトラフィック制限システムと典型的な利用方法を説明する．

2.1. アーキテクチャ

提案するトラフィック制限システム (GateKeeper) は Linux のユーザ空間で動作する双方向ブリッジプログラムである．図 1 は，GateKeeper の構成を示す．簡単のため，図には片方向の中継処理だけを描いた．GateKeeper は普通，外部ネットワークとのゲートウェイルー

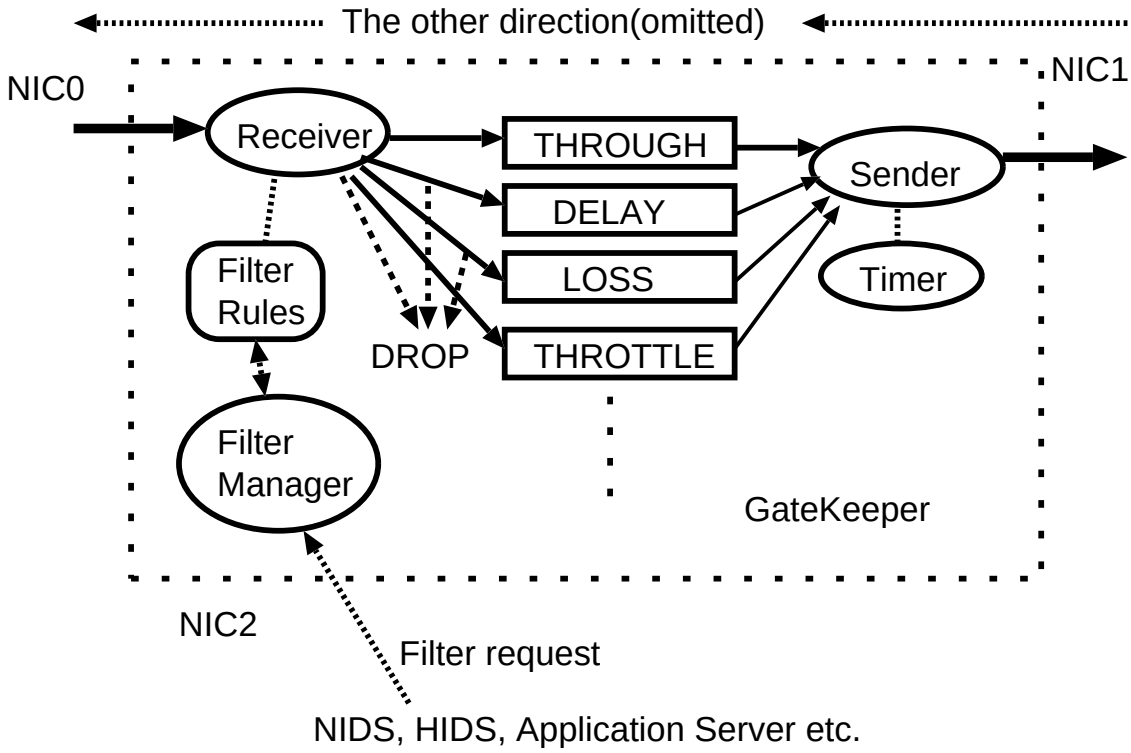

図 1: System overview

タと LAN の間にデータリンク層中継機 (ブリッジ) として挿入設置する．故障した場合は，LAN ケーブルを差し替えて GateKeeper をバイパスすることで，通信を回復することができる．本システムは，実時間クロックタイマ (timer)，フィルタマネージャ(filter manager)，双方向の送信 (sender) と受信 (receiver)，の合計 6 つのスレッドから構成される．

受信スレッドは，ネットワークインタフェースから，入力フレームを受け取り，IP ヘッダとプロトコル番号，ICMP の場合タイプ，コード UDP/TCP の場合ポートで指定したフィルタルールと比較する．一致するルールがなければ，フレームは送信スレッドによって，すぐ反対側のネットワークインタフェースから送信される．一致するルールがあれば，受信したフレームは，そのルールに従い，DELAY，LOSS，または THROTTLE 用のカスタムキューに渡されるか，破棄 (DROP) される．送信スレッドは，送信すべきフレームがキューに溜っていないか短い間隔で定期的に調べ，あれば，送信する．

フィルタマネージャのスレッドは，他のホストからのフィルタルールの更新要求を受け付けるとともに，有効期限等のあるフィルタールールを管理する．SMTP や Web のようなアプリケーションサーバは，自ホストを守るためのフィルタルールの更新を TCP または UDP 通信でフィルタマネージャに要求する．

IDS からの警告は，データベースに保存され，フィルタマネージャはそのデータベースの更新を定期的に調べ，ルールを変更する．本研究には IDS 自体の開発は含まれない．実験

には，Snort[6] IDS で発生した警告を用いる．

2.2. フィルタと制限キュー

ひとつのフィルタルールの主要なフィールドは (srcIP, dstIP, protocol, port 範囲, 使用回数 (use count), action, action のパラメータ) である．action は，THROUGH, DELAY, THROTTLE, LOSS, DROP のいずれかである．ICMP の場合は，type, code を port 範囲の変数に格納する．THROUGH, DELAY, LOSS に対しては各 1 つのカスタムキューを用意する．THROTTLE に対してはフロー毎にカスタムキューを用意する．DROP に対しては破棄するのでキューは必要ない．詳細は以下の通りである．キューは各方向に用意する．

- **THROUGH:** 一致するルールがない場合，フレームをすぐ送信する．送信処理の遅れで失われないように普通の FIFO キューに入れる．
- **DELAY:** 応用層通信を遅くするために，1 秒あるいはそれ以上の遅延を加える．出発予定時刻順に並べるカスタムキューに入れる．
- **LOSS:** 速すぎる TCP データ転送あるいは UDP，ICMP での送信レートが高すぎるフローをランダムフレーム損失させる．損失の対象にならなかったフレームは FIFO キューに入れる．
- **DROP:** 攻撃と確定したフローのフレームを破棄する．
- **THROTTLE:** 転送率が高すぎるトラフィックを制限する．フロー毎に帯域を設定したカスタムキューを用意する．送信間隔は，各フレームのデータ長と設定帯域で計算する．

THROTTLE と LOSS は，任意の攻撃を遅くする効果があるが，DELAY は，特に，自動化された UBE(Unsolicited bulk e-mail) とパスワード推測による login の試みなどの TCP 通信を抑制する効果があると予想される．ルールの action は，そのルールの利用回数他の情報を用いて状態に依存した更新することも考えられる．例えば，色々なタイプの攻撃が続く場合には，以下のような，action の状態遷移は適切であろう．

- **Flood ping:** DELAY → THROTTLE → LOSS → DROP
- **Port scans:** DELAY → LOSS → DROP
- **Password guessing:** DELAY → DROP
- **UBE:** DELAY → DROP
- **Default:** DELAY → THROTTLE → LOSS → DROP

これらの状態遷移が適切であるかの確認は今後の課題である．

3. 実現

本節では，GateKeeper の実現の詳細について論じる．本システムは C++で記述された 2000 行以下の小さいソフトウェアである．オペレーティングシステム (OS) に依存するのは，データリンク層ソケットと Linux 固有の CMOS クロックを用いた実時間クロック (Linux RTC) である．スレッドは Linux POSIX スレッドを GNU Telephony[2] に含まれる Thread クラスライブラリを経由して用いる．データリンク層ソケットと高解像度タイマが利用できる他のオペレーティングシステムでも若干の修正で実行できる．高解像度タイマがない場合は，オーディオ入力の時間待ちを利用することもできる．

3.1. フレーム入出力

近年の OS には DHCP クライアント等で使用されるので，OS 固有のデータリンク層ソケット機能が含まれる．OS に依存しないデータリンク層入力方法は，tcpdump で使用されているパケットキャプチャライブラリ libPcap[7] であり，それは OS 固有の処理を隠蔽して使いやすくしている．出力機能は最近実現されたがあまり使われていない．本研究ではキャプチャするフレームを選択する必要がないので，データリンク層ソケットを用いる．フレームの分類/選択は，GateKeeper 内部で実現する．

データリンク層ソケットを用いるときの注意点は，あるネットワークインタフェース (NIC) について 2 つソケットを作り，片方から受信して他方に出力すると再度受信され，ループが生じることである．したがって，2NIC のブリッジの場合，ソケットは各 NIC に 1 つずつとしなければならない．図 2 に，データリンク層ソケットの利用方法についてコードの一部を示す．Linux の場合，データリンク層ソケットの場合，protocol family は，PF_INET(イン

```
 1
 2    struct ifreq if_r0, if_r1;
 3    struct sockaddr_ll eth0, eth1;
 4    int len;
 5    unsigned char frame[1600]; // includes datalink layer header
 6
 7    sockfd0 = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL);
 8
 9    memset(&if_r0, 0, sizeof(if_r0));
10    strcpy(if_r0.ifr_name, "eth0");
11    ioctl(sockfd0, SIOCGIFINDEX, &if_r0); // look up interface index from name
12
13    memset(&eth0,0, sizeof(eth0));
14    eth0.sll_family = AF_PACKET;
15    eth0.sll_protocol = htons(ETH_P_ALL);
16    eth0.sll_ifindex = if_r0.ifr_ifindex;
17
18    bind(sockfd0, (struct sockaddr *) &eth0, sizeof(eth0));
19
20    ioctl(sockfd0, SIOCGIFFLAGS, &if_r0); // get current interface setting
21    if_r0.ifr_flags = if_r0.ifr_flags | IFF_PROMISC; // add promiscuous
22    ioctl(sockfd0, SIOCSIFFLAGS, &if_r0); // set promiscuous mode
23
24    //
25    // similar lines for eth1
26
27    // receive a frame from bound if (eth0) (actually called in a thread)
28    len = recvfrom(sockfd0, frame, sizeof(buff), 0, NULL, NULL);
29
30    // send a frame via eth1 (actually called in another thread)
31    sendto(sockfd1, frame, len, 0, (struct sockaddr *) &eth1, sizeof(eth1));
32
33
```

図 2: Datalink layer socket

ターネットプロトコル) でなく，PF_PACKET を用いる．インタフェース識別子を調べたあと，ソケットは特定のインタフェースと関連づけ (bind)，自分宛でないフレームも受信するように promiscuos モードに設定する．フレーム入出力は UDP の場合と同様に，それぞれ，recvfrom()，sendto() を用いる．

3.2. キュー

図 3 に示すようにキューは Frame (フレームを表すオブジェクト) の双方向連結リストで実現する．ただし，到着順 (FIFO) でなく，出発予定時刻順に並べる．

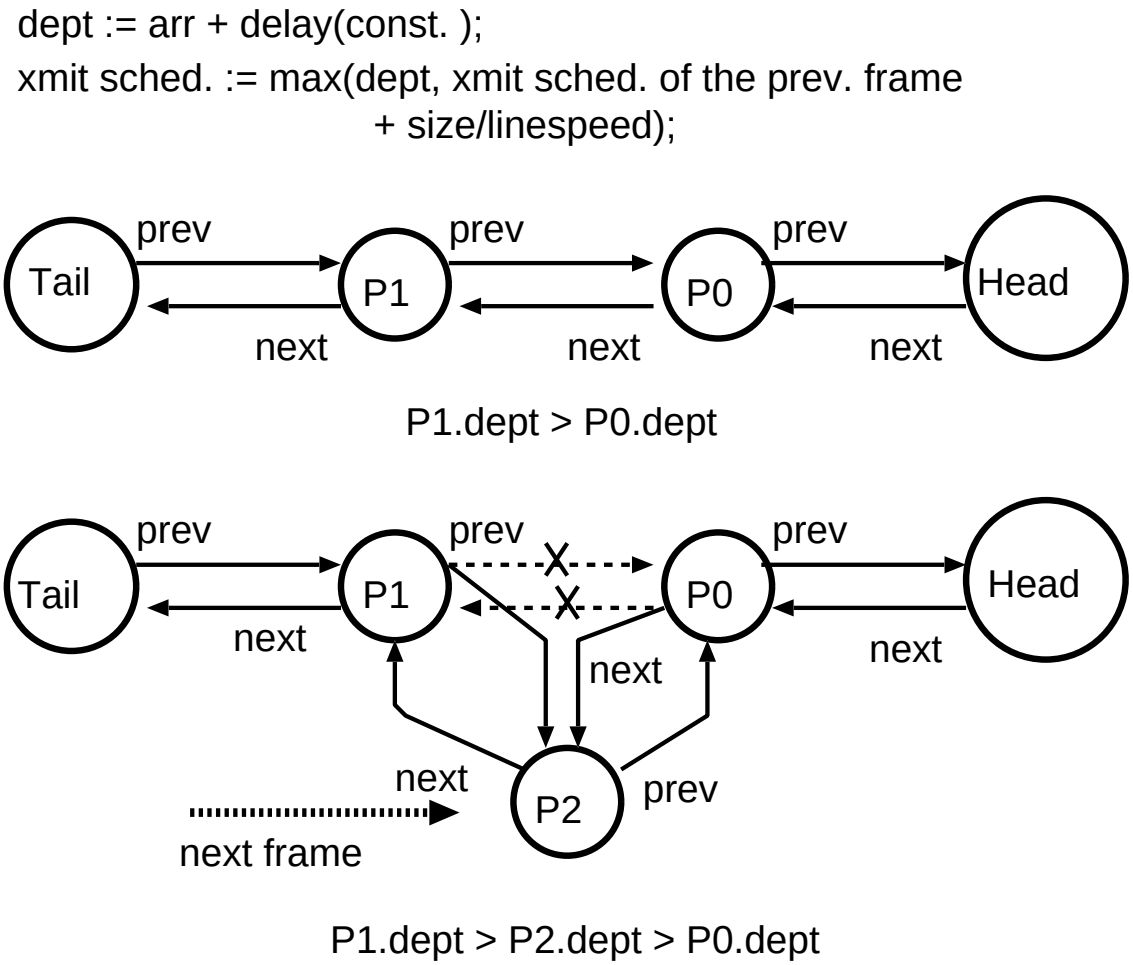

図 3: Frame queue in departure time order

Frame は，フレームのデータ，到着 (受信) 時刻，出発予定時刻と他の必要な情報を含む．LOSS の場合には，到着フレームは THROUGH キューに入れるか破棄される．DELAY の場合には，出発予定時刻は，到着時刻とフィルタルールで設定された一定遅延時刻の和になる．THROTTLE は，送信間隔を各フレーム長と制限バンド幅で計算した時間とすることで模倣する．各キューには最大長 (バイト) が設定され，それを超えたときフレームは破棄される．詳細は，[3] を参照のこと．

3.3. アプリケーションサーバからのフィルタルールの更新

SMTP，Web 等のアプリケーションサーバは GateKeeper の Filter Manager に TCP か UDP でフィルタ更新コマンドを送る．図 4 にその例を示した．

```
Format:

IN/OUT|srcIP|masklen|dstIP|masklen|
     protocol#(0:any, 1:ICMP, 6:UDP, 17:TCP)|
     srcPort(0:any)|dstPort(0:any)|
     DROP/LOSS/DELAY/THROTTLE|
     params for LOSS(frac),DELAY(msec),THROTTLE(Mbps)|
     expiration(sec)|comments(text)|

Example:

     IN|192.168.0.250|32|192.168.0.252|32|1|0|0|DELAY|5000.0|30|ICMP test|
```

図 4: Filter request command from server

コマンド書式は，方向, SrcIP, DstIP, protocol, srcPort, dstPort またはICMP type と code, 制限のタイプ, 制限パラメータ, 有効期限, コメントの順である．ルールの削除，修正，一覧の機能も実現した．

3.4. IDS アラートによるフィルタルールの更新

図 5 は，IDS と GateKeeper の関係を示す．外部ホストが Snort IDS を実行し，その警告がそのホストのデータベースに格納される．GateKeeper の Filter Manager は，例えば 1 秒間隔で定期的にデータベースに新しい警告があるか問い合わせて，あれば対応するフィルタルールを設定する．

図 6 は，データベースに格納された IDS 警告の例である．‘id’ と ‘signature’ はそれぞれ，シリアル番号とアタックのタイプを意味する．32bit の IPv4 アドレスは 10 進数で表現されている．

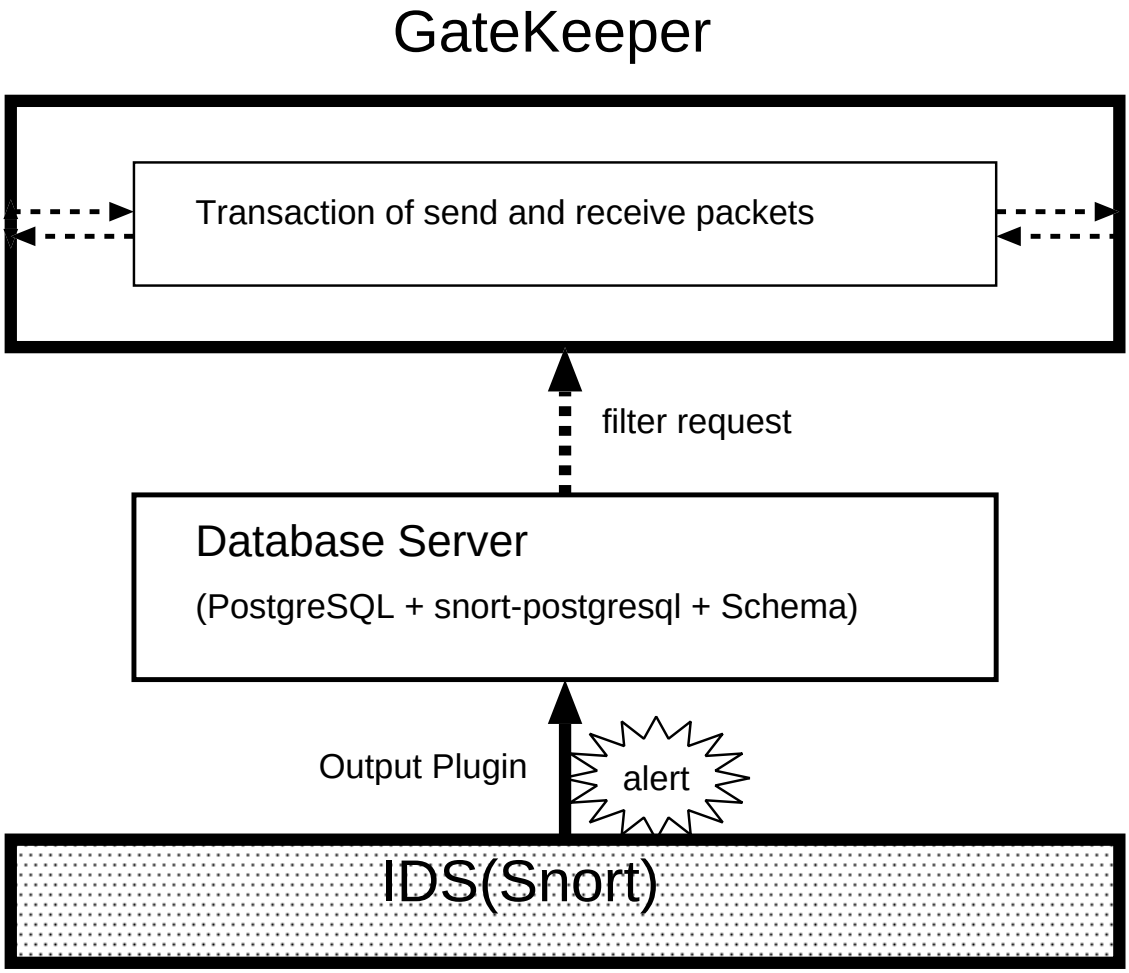

図 5: IDS alert and GateKeeper

```
id: 5027
signature: "TCP portscan"
timestamp: 2006-11-27 12:45:59.436+09
protocol: 6
ipsrc: 3232236033
ipdst: 3232236034
tcpsport: 37593
tcpdport: 161
```

図 6: Alert example

図 7 は，File Manager 内で保持されているルールの例である．フィルタルールは IP アドレスの範囲，ルール設定時刻，最後に使用された時刻，参照回数などを含み，最後の 3 つは，古いルールの削除，action の遷移に使われる．例えば参照回数が多い場合，action を DROP に変更するなどのルール修正が考えられる．

4. 実験結果

本節では，主要な実験結果を述べる．

```
ip_source/mask: AAA.BBB.CCC.123/32
ip_dest/mask:   XXX.YYY.ZZZ.0/24
protocol: 1 (ICMP)
icmp_type/code: 8 (ICMP echo request)
installed: (timestamp)
last_used: (timestamp)
use_count: 100
action: THROTTLE
```

図 7: Filter rule example (ICMP)

4.1. ブリッジ性能

図 8 は，ブリッジとしての往復のフレーム転送性能を示す．実行環境は以下の通りである．

- HostF: CPU Intel Xeon(Dual Core) 3.0GHz, NIC(1000Base-T PCI-Express) Intel 82544EI x 2
- HostS: (比較用) Intel Pentium D(Dual Core) 2.8GHz, NIC(1000Base-T PCI) Realtek RTL8169

UDP スループット上限は GateKeeper をはさんで配置した 2 台の Intel 92540EM(1000Base-T) の間で，iperf[5] の UDP モードで測定した．

図 8 の DIRECT は GateKeeper なしで直結した 2 台のホスト間のスループットで，GateKeeper(HostF) を通過した場合のスループットとの比較するための値である．THROUGH は，GateKeeper において何も制限を設定しないときの最大スループットである．

本システムはユーザ空間のアプリケーションなので，カーネルのブリッジ機能より遅いことが予想される．しかし，THROUGH の線は 700Mbps まで DIRECT と近いことから本システムは 1000Base-T ネットワークにおいては，正常な通信を妨げないスループット性能を持つといえる．HostS の場合は最大スループットが 200Mbps 程度であることから，高速ネットワークインタフェースカード (NIC) を選択する必要があることがわかる．なお，直接接続 (DIRECT) の場合と比較して，THROUGH では 70 マイクロ秒の遅延がある．

4.2. 動的制限変更の効果

図 9 は，動的な制限変更の効果を示す．効果は ping コマンドで 64 バイト (パケット長 84) の ICMP echo 要求を 0.02 秒間隔で送出して計測した．制限は一方向に，素通し (THROUGH)，1 秒の遅延 (DELAY)，100bps の帯域制限 (THROTTLE)，最後に破棄 (DROP) の順で手作業で設定を変更した．

図 9 が示すように，THROTTLE を適用すると往復時間 (RTT) が期待通り線形に増加する．1 秒の DELAY は実験開始から 20 秒から 30 秒の間適用し，設定とおりの遅延が発生したことがわかる．

4.3. 制限の TCP，UDP に及ぼす効果

表 1 は，HostS で DELAY，THROTTLE，LOSS が TCP と UDP スループットに及ぼす効果を示す．TCP スループットは外部ネットワークから内部ネットワークへの iperf を用いたファイル転送で計測した．TCP Window サイズには default 値を用いた．UDP スループットも同様に iperf で計測した．なお，UDP に対する DELAY と LOSS には興味ある結果がないので省略した．UDP スループットに DELAY 影響をあたえないし，LOSS は設定した通りの損失が発生するだけである．

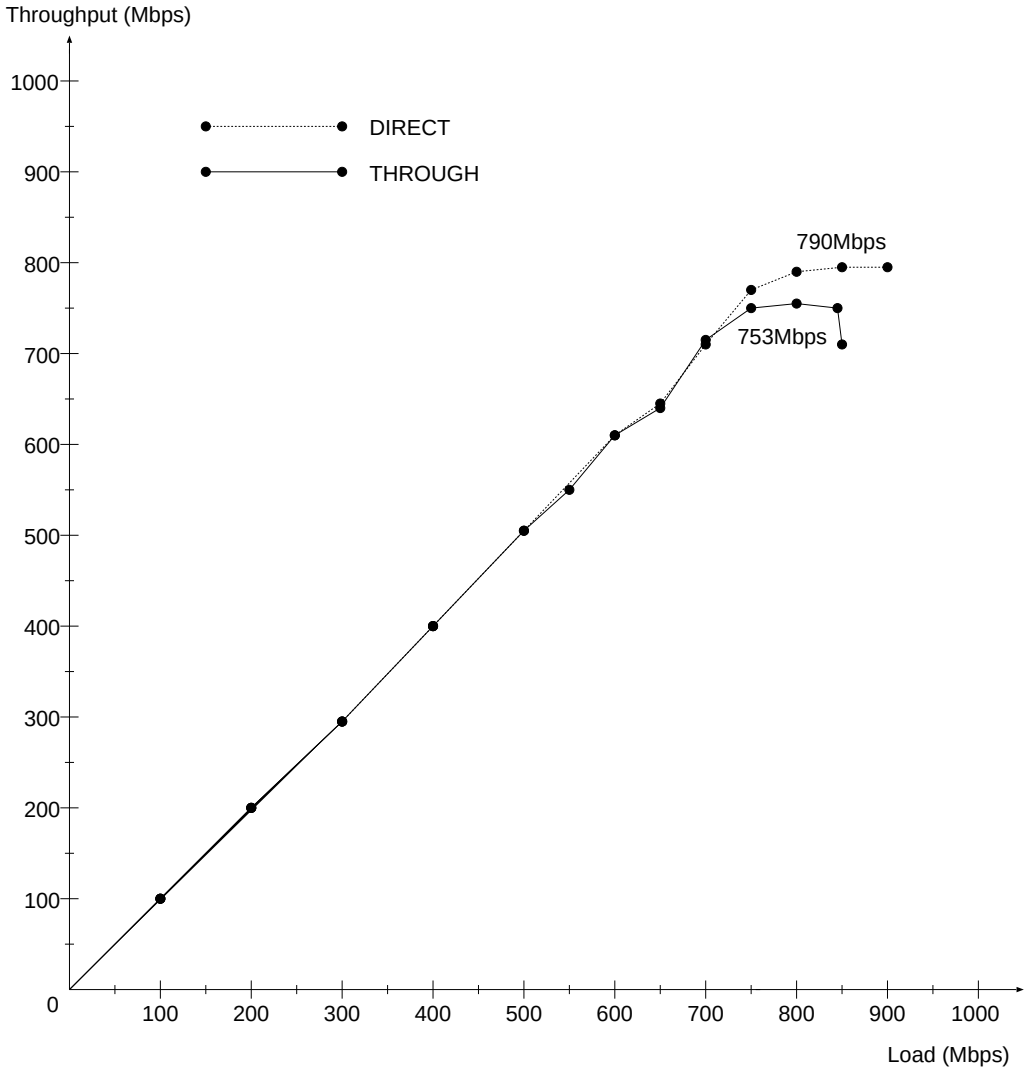

図 8: UDP throughput

表 1 に示されたように，TCP に対しては，どちらかの向きに少し遅延を加えるだけで，スループット抑制効果がある．遅延を加える通信の方向は，GateKeeper のメモリ消費からすると，ACK が戻る方向，すなわち内向きが望ましい．LOSS はデータ送信方向に加えたほうが ACK 方向より効果的である．

THOROTTLE は期待通りの傾向で動作するが，結果のスループットは設定パラメータより 5 から 10%低い．これは GateKeeper の処理オーバーヘッドに起因すると思われる．

4.4. Flood ICMP エコー要求に対する Inbound スロットル

図 10 は，外部から内向きへの高速 ping 実行に対する 10kbps の THROTTLE の効果を示す．0.01 秒間隔で 64 バイトを送信した．フレームは IP ヘッダ，フレームヘッダを含め 94 バイトなので，送信レートは，78.4kbps となる．10kbps の制限により，往復時間は，1.8 秒に増加し，queue 長 150,000 バイトの制限で，ほとんどが破棄される．

4.5. Web サーバ DoS アタックに対する Inbound パケット損失と遅延

HTTP の応用層通信は比較的短いクライアントからの要求と長い応答の繰り返しである．THROTTLE が効果的であることは，すでに述べたので，ここでは LOSS と DELAY の効

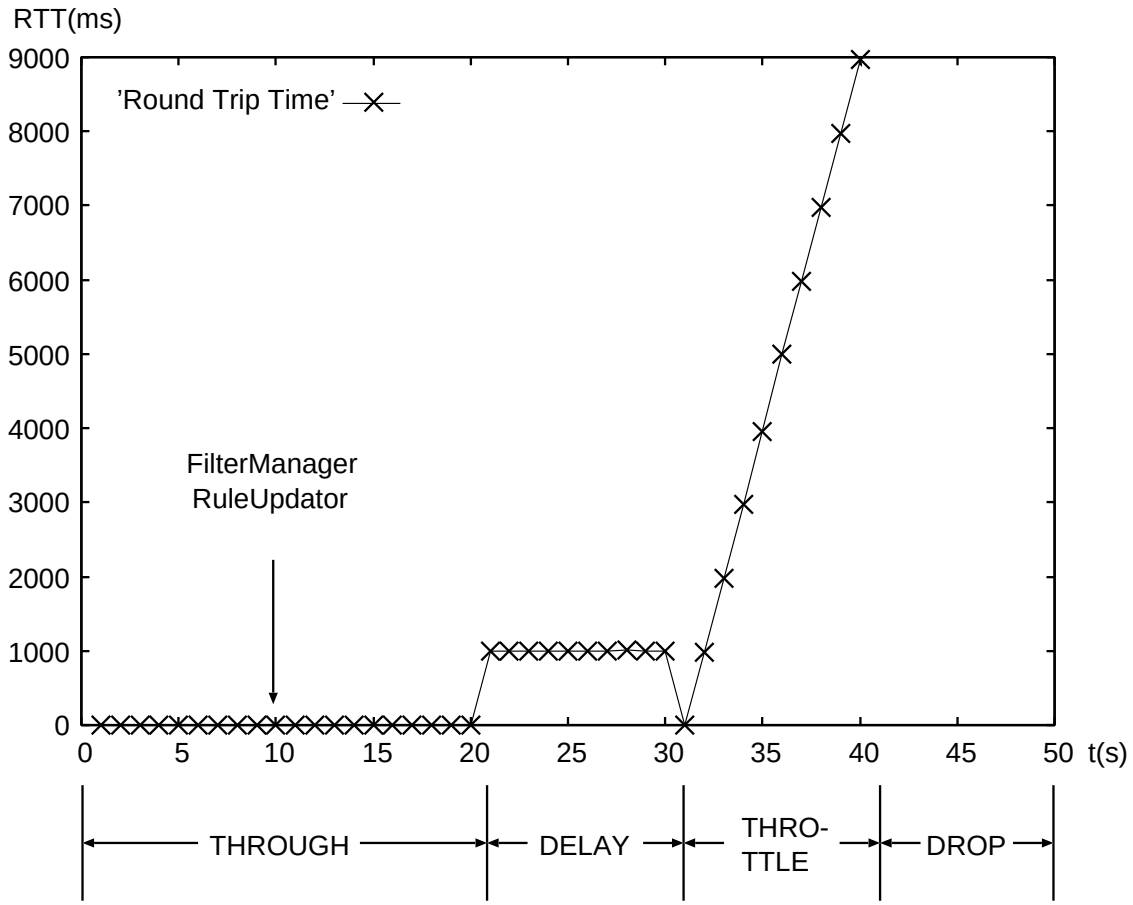

図 9: Effect of limiter measured with RTT

```
ping 192.168.0.250 -i 0.01

64 bytes from 192.168.0.250: icmp_seq=26 ttl=64 time=1528 ms
64 bytes from 192.168.0.250: icmp_seq=27 ttl=64 time=1596 ms
64 bytes from 192.168.0.250: icmp_seq=28 ttl=64 time=1661 ms
64 bytes from 192.168.0.250: icmp_seq=29 ttl=64 time=1728 ms
64 bytes from 192.168.0.250: icmp_seq=30 ttl=64 time=1789 ms

--- 192.168.0.250 ping statistics ---
165 packets transmitted, 30 received, 81% packet loss, time 2284ms
rtt min/avg/max/mdev = 1.050/881.979/1789.819/529.204 ms, pipe 135
```

図 10: Effect of inbound THROTTLE to flood ping

果を調べた．前節の内向きの TCP ファイル転送の場合，外向きすなわち，ACK 方向への LOSS がより有効であることがわかった．しかし，ここでは内部に配置された Web サーバが外部にファイルを送信することを想定するので，外向きに LOSS を設定すると Web サーバの負荷が再送を繰り返すために上昇する．したがって，ここでは内向きの LOSS を設定する．

表 1: TCP and UDP throughput

Type	Param	iperf(TCP, Mbps)	iperf(UDP, Mbps)
(with fast NIC)			
DIRECT	–	520	770
THROUGH	–	496	750
(with slower NIC)			
DIRECT	–	140	220
THROUGH	–	155	204
(with slower NIC)			
OUT DELAY	0	155	—
(ACK, msec)	3	157	—
	10	56	—
	100	7	—
IN DELAY	0	166	—
(DATA, msec)	3	157	—
	5	109	—
	10	59	—
	100	6	—
OUT LOSS	0.1	152	—
(ACK)	0.2	135	—
	0.3	136	—
	0.4	71	—
IN LOSS	0.01	140	—
(DATA)	0.02	90	—
	0.03	37	—
	0.04	20	—
	0.05	10	—
	0.1	1	—
IN THROTTLE	150	135	143
(DATA, Mbps)	100	95	95
	50	48	49
	10	9.6	9.7
	5	4.9	4.9
	1	1.0	1.0

表 2 は Web サーバへの擬似 DoS 攻撃に対する LOSS と DELAY の効果を示す．擬似攻撃は，1MB のファイルを並行して 100 転送要求することとし，apache に付属するベンチマーク用プログラム ab を，“ab -c 100 -n 100 http://server/1MB.dat”，で実行した．

“local” の 550Mbps は，Web サーバホスト内での転送性能で，Web サーバの性能限界を意味する．外部クライアントからの最大スループットは 189Mbps である．内向きの 0.25 以上の LOSS は，スループットを著しく減少させる．これは，クライアントにパケットロスが

表 2: Loss effect to 100 concurrent Web server accesses for 1Mbyte file

IN Loss	tput(Mbps)	IN Delay(ms)	tput(Mbps)
local(max)	550	—	—
0.00	189	0	—
0.10	193	10	176
0.20	166	100	125
0.25	65	200	69
0.30	29	300	56
0.35	8	500	33

多いと TCP 接続と HTTP 要求を繰り返す必要が生じるからである．また，200 ミリ秒以上の遅延もスループットを著しく減少させる．これは，往復時間が増加し TCP のデータ転送率が低下するからである．

4.6. SMTP サーバ UBE 転送に対する長い遅延

TCP 接続成功のあと，SMTP セッションは，挨拶 (HELO)，エンベロープアドレスの指定 (MAIL FROM:，RCPT TO:)，そしてメッセージヘッダと本文の送信 (DATA) の順に進む．メッセージが短ければ，THROTTLE または LOSS は，長い DELAY より効果が低い．ほとんどの UBE(Unsolicited Bulk Email) メイラーや迷惑メイルをまきちらすウイルスは，最初の挨拶または途中のコマンド送信に対する応答が数秒遅れると，メッセージ送信をあきらめる．サーバにおいて故意に応答を遅らせることは ‘tarpit’ と呼ばれる．いくつかの SMTP サーバにおける spam 対策，login 手順におけるパスワードアタック対策として実現されている．

GateKeeper で，tarpit 機能を持たない SMTP サーバ等のかわりに tarpit 機能を実現できる．例えば，各パケットに n 秒の遅延をあたえると, セッション全体で少なくとも $7n$ 秒の合計応答遅延が生じ，あきらめが悪い UBE メイラにも効果があると期待できる．

表 3: Delay effect to 8KB 100 e-mail transfer (10 to 20 concurrent)

Delay(sec)	IN Time(sec)	OUT Time(sec)
0	12	12
1	41	43
2	85	81
3	119	124
4	163	165
5	201	205

表 3 は，2 つの SMTP サーバ間での擬似 UBE 転送に対する DELAY の効果を合計転送時間で示す．時間は SMTP サーバの転送記録で求めた．外部 SMTP サーバ (postfix) に 1 秒以内に 8KB のメッセージを 100 通転送要求し，外部サーバが内部 SMTP サーバ (postfix) にこれらを転送するが，その間に GateKeeper で通信制限を施したところ，20 以上の並行転送がみられた．表に示されたとおり，長時間の遅延は SMTP セッションの進行速度を著しく低

下させる効果がある．内向き，外向きの遅延の効果は同様であり，3 から 5 秒程度の遅延が spam 対策として有効である．

5. まとめ

本研究で提案したトラフィック制限システムが設計とおりに動作することを示した．ユーザ空間のブリッジプログラムとして，カーネルブリッジの限界に近いスループットを得た．また，独自プロトコルでアプリケーションサーバ等からの指示を受けるか，別に設置した既存 IDS のアラートデータベースから自動指示が可能となった．

トラフィック制限の効果は，単純なポートスキャン，ICMP flooding，Web サーバ DoS, SMTP サーバへの UBE で検証した．帯域制限は，一般的に転送レートを制限することに効果があることを示した．TCP データ転送方向の低い率の損失，または 10 ミリ秒程度の小さい遅延はデータ転送率を下げることに効果があることを示した．内向きの高い率の損失，どちらかの向きの大きな遅延は Web サーバ，SMTP サーバへの並行セッションの速度を下げることにそれぞれ効果がある．

本システムは制限ルールの自動更新の開発，簡便な侵入予防システム (IPS) の一部としての使用に有用である．

今後の課題は，より多くの擬似アタックの実験，実ネットワークにおける実験，制限ルールの自動更新，などである．さらに，外部からのルール更新のための通信の認証と暗号化，あやしい通信のハニーポットへの転送機能の実現が望ましい．通信開始時からのハニーポットへの切替えは簡単であるが，TCP の接続時セッション途中での切替えが難しいと思われる．

参考文献

[1] Aoyama M., Kojima M., and Goto K., *Design and Implementation of a Traffic Limiter for Network Security*, Proceedings of the 16th International Conference on Systems Science (ICSS07), Wroclaw, Poland, Vol. II, pp. 213–220, 2007.

[2] GNU Telephony, *GNU Telephony Libraries*, http://www.gnu.org/software/commoncpp/, http://sourceforge.net/projects/gnutelephony/ (accessed May, 2007).

[3] Ihara A., Murase S., and Goto K., *IPv4/v6 Network Emulator using Divert Socket*, Proceedings of the 18th International Conference on Systems Engineering (ICSE2006), Coventry, UK, pp. 159–166, 2006.

[4] Intoto Inc., *IntruPro inline IPS*, http://www.intrupro.com/.

[5] NLANR Distributed Applications Support Team, *Iperf The TCP/UDP Bandwidth Measurement Tool*, http://dast.nlanr.net/Projects/Iperf/ (accessed May. 2007).

[6] Snort, *Snort Web Page*, http://www.snort.org/ (accessed May, 2007).

[7] Tcpdump/libpcap, *Tcpdump/libpcap Public Repository* http://www.tcpdump.org/ (accessed May, 2007).