ネットワーク障害の模倣を目的とする仮想ルータの設計と実現

2001MT031 伊原 亜希　　2001MT070 村瀬 進哉
指導教員　後藤 邦夫

1　はじめに

現在，急速なインターネットの普及に伴い，情報がデジタル化されインターネット上で扱われることが当り前になってきている．

インターネットにおける通信の品質は接続形態や中継経路によって異なる．自己の管理下にないインターネットでスループットや遅れを自在に制御することは不可能であり、また大規模な実験ネットワークを用意することも困難である．ネットワーク障害を模倣する既存のソフトウェアはいくつか存在するが，IPv6 で使用できるものはない．

過去に伊藤らが行った研究 [1] では，IPv4 における仮想ネットワークを構築し，サーバで送られたパケットに対して故意に通信障害を起こし性能評価を行っていた．しかしこの研究では，複数障害の場合にプログラムで指定する遅延時間と実際に発生する遅延時間に若干の差が出てしまい，遅延障害での正確な性能評価が出来なかったと考えられる．さらに IPv6 に対応しておらず，QoS(Quality of Service) 制御などの IPv6 の利便性を模倣できていない．

そこで本研究では，IPv4 だけでなく IPv6 でも，障害を発生する仮想ルータプログラムを作成し，1 つのホストだけで簡単に広域インターネットを模倣できる実験ネットワークの構築を可能とした．Dummynet, NISTNET[2] などのカーネル空間での障害模倣と比較すると最高パケット転送 (forwarding) 能力は低いが，IPv6 機能があること，プログラム追加変更が容易であることから，中低速ネットワークの模倣においては，既存の障害模倣機構より優れていると言える．

今年度の研究では

1. IPv6 における Divert Socket 機能実現のためのカーネルコード作成
2. 昨年度のプログラムを参考にし，障害プログラム・単一ルータプログラムの作成
3. 複合障害の模倣
4. 模倣した障害を用いた QoS 制御

を進める．

伊原は主にネットワーク構築・カーネル再構築・確率分布を，村瀬は主に IPv6 関連・Divert Socket・障害プログラム・QoS を担当する．

2　仮想ネットワークについて

本節では，本研究で構成する仮想ネットワークについて説明する．

2.1　ネットワークモデル

本研究では，1 台の仮想ルータと 2 台のホストの計 3 台の PC から構成される仮想ネットワークで実験を行う． 仮想ルータの OS には昨年度同様に FreeBSD を利用し，kernel 機能である Divert Socket によってネットワーク障害の模倣を試みる．ホストには Linux を利用し，2 台の PC 間で通信し，評価する．図 1 に仮想ネットワークモデルを示す．

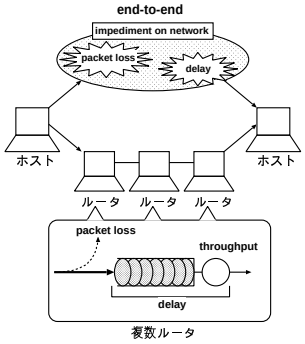

図 1: 仮想ネットワークモデル

2.2　Divert Socket

Divert Socket とは OS である FreeBSD で標準搭載されており，IP 層での IP パケットの横取り・再注入をする事ができる socket である．一般的には NAT 機能をユーザプロセスで実現する natd などで使用されている．ユーザプロセスでパケットを自由に操作できるので，性能評価をするためのプログラム作成が容易である．また ipfw で指定したルール毎に最大 65536 ポートの divert socket に振り分け可能というメリットがある．パケットの通過や破棄を行うための IP ファイヤーウォールにパケットを Divert Socket へ送るルールを追加すると，指定したポートでユーザプログラムから入出力できるようになる．

3　IPv6 DIVERT 実装

現在，FreeBSD の IPv6 処理には Divert Socket 機能がないので (カーネルバージョン 5.2.1)，新たにカーネルコードと socket アプリケーション (障害プログラム) を作成した．

OS 全体のファイルやオプションのリストを更新するために，/usr/src/sys/conf/options に IP6DIVERT を追加し，/usr/src/sys/conf/files に netinet6/ip6_divert.c optional ip6divert を追加した．そして IPv6 のファイヤーウォールと Divert Socket を使用できるようにするため，カーネルに

```
options     IPV6FIREWALL
options     IPV6FIREWALL_VERBOSE
options     IPV6FIREWALL_VERBOSE_LIMIT=100
options     IP6DIVERT
```

を加えてカーネルの再構築を行った．
次に，Divert 機能の概要を図 2 に示す．

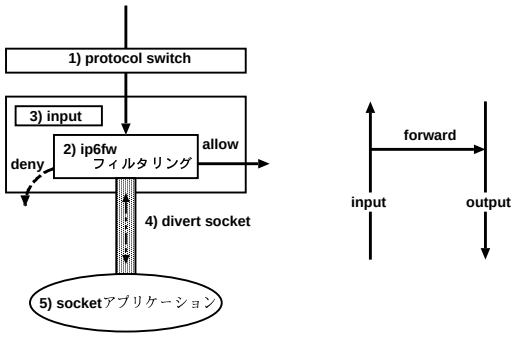

図 2: Divert 機能の概要

Divert 機能を IPv6 に追加するため，以下 5 点の作成・修正を行った．
1) プロトコル別の処理関数を定義した protocol switch に，プロトコル'IP6_DIVERT' のソケット処理関数群を追加した．
2) IP6FW のカーネルコードに IP6_DIVERT のフィルタを追加した．またカーネルのフィルタを設定する ip6fw コマンドに DIVERT オプションを追加した．
3) ip6_input.c などでは，修正した ip6fw の機能を用いて，DIVERT 設定ルールに適合したパケットを作成した ip6_divert.c 内の関数に渡す．
4) ip6_divert.c を変更し，ip6fw のルール番号を認識して，各ルールに該当するパケットを配送予定リストに加える．Divert を通過したパケットには，それとわかるタグを付ける．
5) socket アプリケーションでは，IPv6 パケットを処理するために socket(PF_INET6，SOCK_RAW，IPPROTO_DIVERT) でソケットを作成し，ip6fw で指定したポート番号をソケットに対応づける．また，recvfrom/sendto でデータの送受信を行う．
また本研究での実験では用いないが，Linux2.4.X 用 IP_DIVERT パッチをもとに IPv6 用 DIVERT 修正カーネル・ip6tables 修正コマンドも用意し，一応の動作を確認した．

4 障害の実装

この節では，各障害の説明と実装方法を説明する．

4.1 end-to-end 通信の模倣

■パケット損失 パケット損失の種類として突然の災害, 中継機器・ハードウェアの物理的な障害によるパケット損失や，回線でのビットエラーや多量のトラフィックが原因で発生するルータでのバッファ溢れから起こるパケット損失等が挙げられる．図 3 に分類を示し，以下に図中の番号で示したものについて説明する．

1. パケット毎に指定した確率で損失する．
2. ビット毎に指定した確率で損失する．これにより損失確率がパケットサイズに依存する．
3. あらかじめ用意したビットエラーパターン・パケットロスパターンを再現する．
4. マルコフ連鎖モデルに従い，パケットが断続的に連続して損失する．

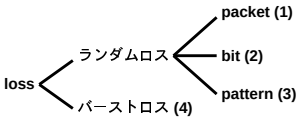

図 3: loss の分類

■通信の遅延 通信遅延の種類には，通信経路上の各ルータが処理に要する事で発生するネットワーク伝送遅延，回線やルータ (複数) の障害による通信遅延等が考えられる．図 4 に分類を示し，以下に図中の番号で示したものについて説明する．

1. 伝搬遅延等で一定の負荷がかかり全てのパケットが同じ時間分だけ遅延する．
2. パケットが確率分布に従った可変の遅延時間分だけ遅延する．可変的な遅延時間によりパケット到着順序が入れ換わる事を考慮しており，確率分布の通りに結果は出力される．
3. 2 とは異なり，順序が変わらないように修正されている．
4. あらかじめ用意されたパターン通りに遅延を発生させる．

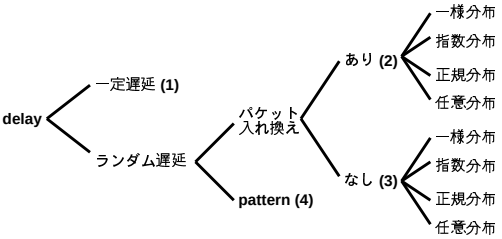

図 4: delay の分類

4.2 単一ルータの模倣

図 1 の複数ルータの模倣として，ルータ障害ユニットを直列に繋いで使用した．各サービスに必要な帯域を確保するためにパケットを格納するキューのサイズを固定し，指定された転送速度でパケットを送り出す状態を模倣するプログラムを作成した．さらにキューイングアルゴリズムを選択することによって複数のパケット配送

方法を選択することのできる QoS 制御機能を模倣した．キューイングアルゴリズムに Round-Robin Queueing, Priority Queueing, Weighted Fair Queueing[3] を選択し，優先度付き TCP パケットの通信速度を観測した．バックグラウンドには優先度の低い大量のトラフィックを流し，パケットサイズを変化させて実験を行った．

4.3 その他

パケット重複の模倣として，ルータがパケットを受け取ったときに同じパケットを何度も受信し送信するプログラムなどが考えられる．

4.4 プログラム実現上の工夫

昨年度の研究ではマルチスレッドを利用することでパケットの送信と受信を独立に行うことによって性能を評価していた．しかし障害プログラムを複数動かすと処理速度が落ちてしまい，遅延を伴うプログラムの性能評価が正しくできなかった．そこで本年度の研究ではシングルスレッドにし，select 関数を利用した．パケットを受信するソケットを選択することが可能となり，処理がスムーズに行われるようになった．

また，遅延の発生に当初使用していた usleep では遅延を可変とする場合にパケットが入れ換わる状態を模倣できなかった．そこで setitimer システムコールを利用することで，パケットの入れ換えを実現することができた．

5 複数障害の発生

昨年度の研究では IP firewall で divert への送受信のルールを用い，1 つのルールで 1 つの障害を起こすことで複数の障害を模倣していた．本研究では，同じホスト上でプロセス同士が効率的に通信する Unix Socket を利用することで模倣した．それにより，プログラム同士を直接接続することが可能になり，無駄なパケット転送時間を省くことが出来て，ルータに余分な負荷が掛からなくなった．さらには直列だけでなく並列に繋ぐことも可能である．複数の障害ユニットを直列に繋ぐことで，end-to-end(複数障害) と複数ルータが表現できる．

6 性能評価

構築した仮想ネットワーク上で，作成した障害プログラムを実行してデータを収集し，性能を評価した．IPv4 では ping コマンド，IPv6 では ping6 コマンドを用いて icmp パケットを流して片道のみ障害を与えて測定した．また，実験で得る値は 1 つの標本にすぎず，実行するたびに少しずつ異なった結果がでる．そこで性能評価に用いる数値には 95%の信頼区間を用いた．実験に用いたコンピュータの spec は Intel Pentium II，500MHz, 320MB であり，100Base-TX の NIC を 2 つ使用した．なお，以下で示すプログラム以外の性能評価については省略する．

6.1 ランダムロス

パケット損失の模倣として，パケット毎に一様分布に従った確率で落とす障害プログラムを作成した．

表 1 に，パケット損失率を 5(%) と 20(%) に設定してプログラムを実行した結果を示す．損失パケット数には 95%の信頼区間を，損失率には算出したパケット数を送信パケット総数 (10000) で割った確率を示す．

表 1: パケット損失率 (%)

設定損失率	IP	損失パケット数	損失率 (%)
5	v4	498.4 ± 24.93	4.98 ± 0.25
	v6	506.3 ± 30.47	5.06 ± 0.30
20	v4	2002.7 ± 28.42	20.0 ± 0.28
	v6	1988.9 ± 69.72	19.9 ± 0.70

実験回数と信頼区間より，期待する値に十分近い結果が得られたと言える．

6.2 バーストロス

パケット損失の模倣として，パケットが連続してロスするプログラムを作成した．図 5 において，状態を i(0:パケット通過，1:パケットロス) としたときの状態確率を P_i，状態 i から j への推移確率を p_{ij} で表すものとする．

このとき，

$$P_0 + P_1 = 1 \tag{1}$$

$$P_0 \times p_{01} = P_1 \times p_{10} \tag{2}$$

式 1，式 2 より，

$$P_0 = \frac{p_{10}}{p_{01} + p_{10}} \tag{3}$$

となる．

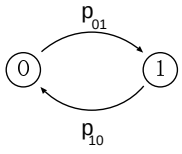

図 5: 状態推移図

次に，パケットが通過した場合は”o”，ロスした場合は”x”で表示し，評価対象とするネットワークの状態を表す．パケット損失率を 80(%) に設定した状態で，推移確率が $p_{01} = 8\%$，$p_{10} = 2\%$ の場合と，$p_{01} = 80\%$，$p_{10} = 20\%$ の場合の結果を以下に示す．連続した 1000 パケットを用いて観測を行った．

$p_{01} = 8$，$p_{10} = 2$ の場合

```
ooooooxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxoo
oooooooooooooooooooooooxxxxxxxxxxxxxxxxxxxxxxxx
```

$p_{01} = 80$，$p_{10} = 20$ の場合

```
oxxxxoooxxxxoxxoxxxxxxxxxoxxxxxxxxxooxooxxxxxxxx
xxxxxxxxoxxxxxxoxxxxxoxxxxxoxoxxxxxxxxoxxxoxxx
oxxxxxoxxxxxxxxxxxxxxxoxxxxxxxxoxxoxxxoxxoxoxxx
xxoxxxxxxxxxxxxxxxxxxxxxxxxxxxoxoxxxxxoxxxxxxxx
```

この結果より，p_{01} が大きくなるほど連続して通過するパケット数は減少し，p_{10} が小さくなるほど連続してロスするパケット数は増加する様子が確認できた．状態確率が等しい場合においても，推移確率が異なると連続して通過またはロスするパケット数が大きく異なる模倣を実現することができた．

6.3 単一ルータの模倣

単一ルータの模倣としてスループットを制限するプログラムを作成した．TCP ではファイル転送プログラムを作成して実験に用いた．また ICMP では ping6 のパケットサイズを 1000(byte) に固定して 100000 パケットを送った．表 2 に実験結果を示す．

表 2: 通信速度 (Mbps)

	TCP	TCP × 2	ICMP
non divert	90.16	90.16	10.23
throughput 100M	5.57	—	2.71
throughput 10M	3.73	3.84	2.65
throughput 1M	0.87	0.89	0.70

Divert Socket を用いた場合, TCP･ICMP 共に通信速度が著しく下がる．また TCP でスループットを 100M から 10M に制限した場合，ICMP に比べて速度が制御されている．これはメモリ管理でパケット到着ごとに malloc をしていることが考えられる．結果より，TCP では上限が 5.57(Mbps)，ICMP では上限が 2.71(Mbps) ということが確認できた．なお，実験結果で期待した上限速度が出ていないのは，プログラム中での処理能力が不完全なためと考えられる．

6.4 複数障害 1(指数分布に従った遅延と一定遅延)

図 6 に，指数分布の平均遅延時間を 100(ms) に，一定遅延を 100(ms) に設定して障害ユニットを直列に実行した結果を示す．横軸には発生した遅延時間 (ms) を，縦軸には発生回数 (回) をとり，実験結果を’ ×’ で，平均を 100(ms) とした指数分布の密度関数を’ ＋’ で表した．また，図は v4 の結果を示すが，v6 も同様の結果になることが確認できた．

結果のグラフは，指数分布の密度関数のグラフを x 軸方向に 100 だけ平行移動したものと重なることから，2 つの障害の模倣することができた．

6.5 複数障害 2(ランダムロスと一定遅延)

表 3 に，パケット損失率を 5(%) と 20(%) に設定してプログラムを直列に実行した結果を示す．損失率には算出したパケット数を送信パケット総数 (10000) で割った確率を，平均遅延には観測された遅延時間を示す．

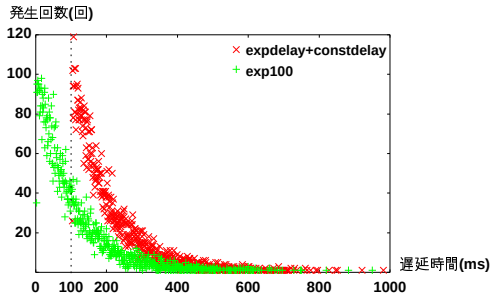

図 6: 平均遅延を 100ms とした場合 (発生回数は 1ms 区間)

表 3: パケット損失率 (%) ＋平均遅延 (ms)

設定損失:遅延時間	損失率	平均遅延
5 : 100	4.94 ± 0.22	102.31 ± 0.01
5 : 1000	5.05 ± 0.20	1002.28 ± 0.03
20 : 100	19.99 ± 0.25	102.31 ± 0.01
20 : 1000	20.05 ± 0.43	1002.27 ± 0.02

結果より，プログラムを単体で実行させた場合と比べて変わらない結果が得られた．遅延時間が約 2(ms) 程度増えているのは，プログラムが増えたことによる処理時間が原因と考えられる．

7 おわりに

本研究では，IPv4 および IPv6 でもネットワーク障害を発生する仮想ルータプログラムを作成した．広域ネットワークを模倣するための実験ネットワークの構築のため，Divert Socket 機能と IPFW を IPv6 に拡張した．また複数障害の模倣では Unix Socket を用いることで負荷が少なくなり，障害ユニットを直列にも並列にも組み合わせることが可能になった．これにより，end-to-end の障害・複数ルータの障害・優先キューの模倣を実現できた．また負荷実験を通して，指定した確率分布に従ったパケット損失・遅延が得られることが確認できた．

しかし，キューイングモデルの種類が少なく複雑な QoS 制御が行うことができなかった点が今後の課題として考えられる．またプログラム throughput 内のメモリ管理では，始めに十分な領域を確保しておきそれを使うようにすることで処理効率を上げられると考える．

参考文献

[1] 伊藤 洋介，中本 拓也，大藤 純一，吉田 秀考：性能評価のための仮想ネットワークの構築，南山大学経営学部情報管理学科卒業論文 (2004.3).

[2] NIST Net Home Page, http://www-x.antd.nist.gov/nistnet，2002.

[3] John Nagle：Congestion control in IP/TCP internetworks，RFC896，1984.