NANZAN-TR-2009-03

仮想ネットワークスタックを用いたIPネットワークエミュレータの設計と実装

浅野洋介　　後藤邦夫　　杉山裕介

November 2009

Technical Report of the Nanzan Academic Society
Information Sciences and Engineering

仮想ネットワークスタックを用いたIPネットワークエミュレータの設計と実装

浅野洋介　　後藤邦夫　　杉山裕介
南山大学 大学院 数理情報研究科

和文概要　本研究では，これまでに開発したインターネットアプリケーションの性能評価やルーティング実験のためのIPv4/v6ネットワークエミュレータの機能を拡張した．

このエミュレータは，Linuxでユーザ空間プロセスとして動作するGNU common C++のThreadといくつかのライブラリクラスを利用したC++プログラムである．

旧版では，1台のPCで，10以上のホストと遅延，帯域，損失のある回線から構成されるIPv4とIPv6のネットワークを模倣できるが，動作環境として改造Linuxカーネルが必要で，本物のルータやホストの機能，例えば，動的ルーティング，アプリケーションサーバ，クライアントの機能がなかった．現在版では，最近のLinuxカーネルの仮想ネットワークスタック機能を利用した仮想ホストにおいて，ルーティングプログラムやサーバプログラムを起動できるので，旧版より現実に近いネットワーク構成が模倣できるようになった．

本研究では，これまでのエミュレータの機能を整理統合し，操作性向上のために，模倣するネットワークのトポロジ保存/読み出し機能，GUIによるトポロジ編集と実行制御機能を追加した．

キーワード: 通信，ネットワークエミュレーション，性能評価，インターネットプロトコル

1. はじめに

広域ネットワークにおけるアプリケーションの性能評価のためにはさまざまなネットワークシミュレータやネットワークエミュレータが使われている．シミュレータはネットワーク設計やネットワークアプリケーション等の性能評価に使われることが多い．一方，エミュレータはホスト内に仮想ネットワークを構築し，実際にパケットを送受信することで実ネットワークにより近い環境を模倣することができる．さらに外部ホストからフレーム (パケット) を横取りし，エミュレータ内で処理して再度，外部ホストへ書き戻すことが可能である．

オープンソースの end-to-end の通信を模倣できるエミュレータとして代表的なものは，Nistnet[5] である．これはリンクエミュレーションのみで，処理対象とするネットワークアドレスのプレフィックス指定がなく，IPv6にも対応していない．またNCTUns[7]では，さまざまなネットワークトポロジを表現できGUIも実装されているが，独自カーネルが必要で，実行環境が整うLinuxディストリビューションが限定され，IPv6にも対応していない．さらにFreeBSDのIMUNES[9]はホストやルータをエミュレーションできるが，独自拡張カーネルの再構築が必要である．

我々が開発してきたGoto's IP Network Emulator(以下，GINE) [1][8]は，多数のルータとリンクで構成される広域ネットワークを模倣することができるネットワークエミュレータである．各リンク毎に与えられた確率分布に従って遅延やパケットロスなどの通信障害を容易にエミュレートすることができ，バンド幅を指定できる．また，C++クラスライブラリを用いたユーザ空間プログラムで記述されているため自由機能が追加でき，IPv6にも対応している．

[1]ではパケット横取りのために独自拡張カーネルを用いる必要があった．[8]では，横取り方法を2.6.14以後の標準カーネルに含まれる方法に変更し，さらに，2.6.26以後の仮想

ネットワークスタックで仮想ホストを表現し，動的ルーティング，エミュレータ内でのアプリケーションサーバの起動が可能となった．しかし，GUI 等の視覚的操作やネットワークトポロジ (オブジェクト) の保存，編集，読出機能がなかった．

本論文ではこれまで改良されてきた GINE の機能を整理，統合し，エミュレータの実用性を高めるため，GUI 機能や案内に沿って操作する機能などを GINE に搭載できるように改良する．そして，これらの機能拡張によって大規模かつ複雑なネットワークトポロジのエミュレーションネットワークを構築できるようにする．企業や研究者が仮想的にネットワークを構築してネットワーク構成を容易に学習することができるようなネットワークエミュレータの開発を目標とする．2 節に GINE のシステムアーキテクチャの概要を，3 節にシステムの実現方法を，4 節に GINE のシステム評価について述べる．

2. システムアーキテクチャ

この節では GINE のシステム構成を説明する．GINE のプログラムはシステムコールを多用するため C++と，GNU common C++[2] クラスライブラリのスレッド機能を使用して記述した．

2.1. リンクエミュレーション

リンクエミュレーションは，NistNet[5] で使用できる機能である．しかしこのエミュレータは IPv6 に対応していない．GINE は IPv4/v6 のリンクエミュレーションや IPv4/v6 アドレス，プロトコル，ポートのフィルタリングが可能である．GINE 内のリンクは一般的なデータリンク層のフレームバッファとして使用される独自 Queue によって実現している．

[1] において，FreeBSD の機能である Divert Socket(フレーム横取り) を Linux で使用できるように改良し組み込むことに成功した．加えて FreeBSD では対応していなかった IPv6 の機能も搭載することができた．これらによってフレーム横取りの性能評価でスループット 750Mbps 以上を，さらにホップ数を増やしても 100Mbps 以上のスループットを実現している．しかし使用するカーネルに Divert 機能を追加しなければならなく，最新のカーネルに追加していくのが困難である．またフレームを横取りするためには iptables を修正する必要がある．そこで [8] では，Linux カーネル 2.6.14 から標準搭載されている Netfilter 機能である NFQUEUE[3] を用いた．これによってカーネルの修正，追加せずに容易にフレームの横取りができるようになった．図 1 にリンクエミュレーションモデル例を示す．

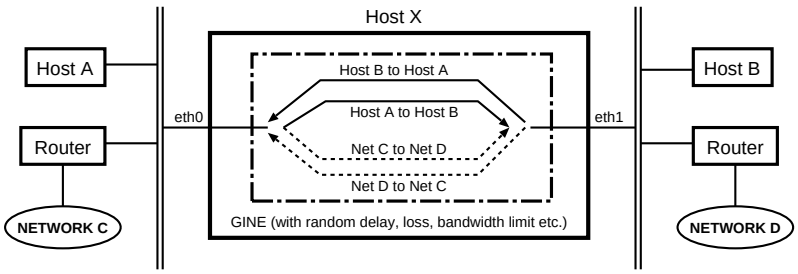

図 1: リンクエミュレーション

例えば Host A から Host B へ送信されたパケットのプロトコルが ICMP である場合のみを NFQUEUE で横取りするように `iptables` でフィルタリングしたいとき，Host X において以下のようにルールを設定する．

```
# /sbin/iptables -A FORWARD -p icmp \
               -s HostA_IPADDR -d HostB_IPADDER -j NFQUEUE --queue-num 0
```

eth0を送信元とするICMPのフレームをFORWARD Chainによって横取りされ，NFQUEUE 0に注入される．NFQUEUEに横取りしたフレームをGINEに注入している．その他のパケットは素通りでHost Bに送信される．ルールにマッチしないフレームであったときは，このNFQUEUEには注入されない．

プロトコル，経路など異なった種類や，往き復りのリンクごとにNFQUEUEを設定すると，違うパラメータの障害をそれぞれで起こすことが可能である．GINEで発生させることができる障害は，ランダム遅延，フレーム損失，リンク容量エミュレーションである．この障害は，独自Queueを用いたリンク上で発生させることができる．

2.2. ホスト/ルータエミュレーション

FreeBSDのIMUNES[9]は，ホストやルータを模倣する機能がある．GINEでもこの機能を使用できるように改良している．

一般的に1台のホストにおいてルータやホストをエミュレーションするためには，次の4つの実現方法がある．

1. 仮想ネットワークインタフェースやポートの使用
2. 仮想化ソフトウェアによる仮想OSの使用
3. スレッド等を用いたユーザプロセスプログラムの作成
4. ネットワークスタックの仮想化

1の方法は，TCPやUDPでのポートを変えることによって最大ポート数のホストをエミュレートすることができる．またtuntap等の仮想デバイスとアプリケーションを接続することでも実現できる．しかしこの方法ではアプリケーション間の通信パラメータ設定や通信の独立性を保つことができない．

2は，User Mode LinuxやVMware，Xenなどの仮想化ソフトウェアを用いる方法である．この場合，ルーティングデーモンを使用することができるが，複数の仮想ホストを同時に使用した場合にはメモリや記憶容量の不足に陥りやすい．

3の方法は，GINEで実装している方法の1つでフレームの横取りや時間計測等で使用している．プログラムですべてを実装しているためルーティングなどの機能も組み込む必要があるが，現在のところ静的なルーティング設定しかできない．複雑なネットワークを模倣するためにはRIPやBGPなどのルーティング機能を実装する必要がある．

そこで，GINEではルーティングデーモンを使用できるようにするためにネットワークスタックの仮想化を採用した．これはルーティングやフォワーディング機能を含めたネットワーク部分を仮想化し，ネットワークホストを構築する方法である．[8]によって仮想化技術であるNetwork Namespace[4]をGINEに導入し，実ホスト，実ルータ等の機能をより容易に実装できるようになった．ネットワークスタックの仮想化は，Linux 2.6.26から標準で搭載されており，カーネル修正は不要であるため容易に使用することができる．ファイルシステムの仮想化も可能であるが，本研究では不要なので使用しない．

GINEでは3と4を組み合わせて利用することでさまざまなネットワークトポロジを構築することができる．図2にGINEによって構築できるネットワークトポロジ例を示す．

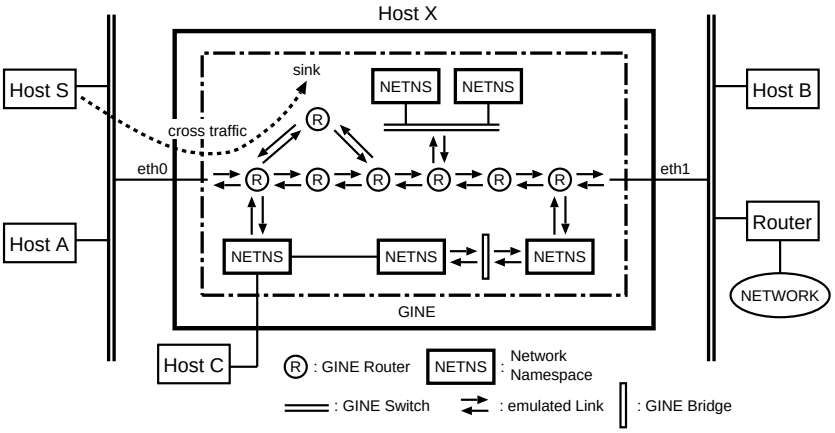

図 2: ネットワークトポロジ

仮想ルータ (GINE Router)，仮想スイッチングハブ (GINE Switch), 仮想ブリッジ (GINE Bridge) などをプログラムによって生成し，さまざまなネットワークトポロジを構成することができる．また仮想ネットワークスタック (以下，NETNS) にルータ/ホストの機能を持たせることでより現実的なネットワーク空間を構築できるようになる．

また流れているトラフィックの合計がネットワークインターフェイスの最大容量より少ない場合に限り，UDP のクロストラフィックを特定のリンク間で与えることができる．

2.3. 改良点の概要

現在までに，GINE にさまざまな機能の追加や改良が行われてきた．しかし機能が増えてきたことで，管理する機能が必要になってきた．本研究では以下の 3 つについて改良と機能追加を行う．

1. 仮想ネットワークスタックの自動生成と端末を用いた管理
2. オブジェクトの保存/編集/読出機能の追加
3. GUI 機能の追加 (ネットワークトポロジの作成/保存/読出)

3. システムの実現

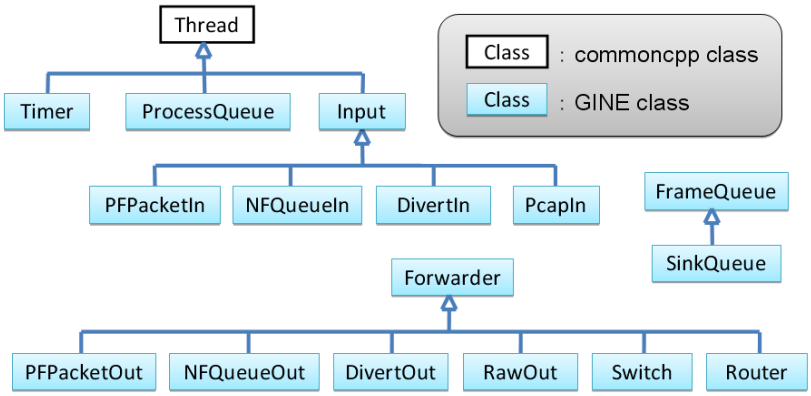

図 3: GINE クラス継承関係

GINE のプログラム構成 (継承関係) は図 3 のようになっており，次の 3 つの主要クラス郡で構成されている．

- common C++の Thread クラスを継承する Timer，Input，ProcessQueue クラス
- パケットを中継または書き戻す Forwarder クラス
- 独自拡張 Queue(FrameQueue) クラス

3.1. タイマを用いたスレッド同期

GINE におけるスレッド同期は，OS に負荷を与えずにタイマによるマルチスレッドの時刻同期を用いて実現している．

タイマには短い時間を測ることができるタイマを用いる．タイマはハードウェアやカーネルに依存するため，以下の 3 種から選択する．

- Linux High Resolution Timer(NANOSLEEP)
  システムの設定と能力に応じて，より低いシステム割り込み負荷で高精度のアラームを実現するタイマ，数マイクロ秒の短時間まで計測可能である．
- Linux Real Time Clock(RTC)
  マザーボード上に実装されている計時専用チップによって時間を計測するタイマで，1/8192 (122 マイクロ) 秒まで計測可能である．
- 44.1kHz のステレオ入力によるタイマ (AUDIO)
  サウンドボード (オーディオデバイス) から 48kHz でサンプリングされたステレオ入力によるタイマである．

フレームの横取りクラスである Input は Thread クラスを継承している．このクラスで生成するオブジェクトは，タイマによって待ち時間を作り，フレームが到着したら独自 Queue に注入している．

Forwarder オブジェクトは Thread で動作していない．以前は Thread クラスを継承して動作していたが，スレッドを用いるとその数が増えるほど CPU 使用率が増加し，通信速度が低下することが多かった．そこで現在の GINE ではそれを防ぐためにパケット入出力で使用されるスレッド数をできるだけ減らすように工夫されている．スレッドの代わりに独自 Queue を Timer で定期的に監視させる ProcessQueue に登録する．そして登録された複数の独自 Queue のフレームが短い間隔で Forwarder に出力される．

3.2. 独自 Queue の実装

エミュレータ内のルータ同士のリンクは図 4 のように GINE のプログラムで生成した独自 Queue へのフレーム入出力によって実現している．

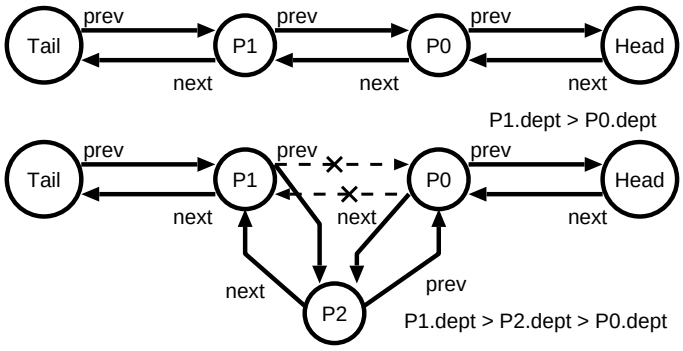

図 4: 独自 Queue の構造

このQueueは出発予定時刻順で形成している双方向リストである．これは短い時間間隔で，ランダム遅延による順序変更によって実装されている．またこのQueueは有限長であり，バンド幅制限，フレーム遅延，損失を模倣することができ，一般的なデータリンク層のフレームバッファとして使用できる．エミュレータ内を流れるフレームの生成はGINEのプログラム内で実装されている．

$$\text{フレームの出発時刻} = \text{到着時刻} + \text{一定またはランダム遅延時間} \tag{3.1}$$

フレームが到着すると出発時間は，到着時間に一定またはランダム遅延だけ追加され(3.1)式のように計算される．そしてフレームは出発時間の順番にこのQueueに入れられる．

高速化のためプログラム起動時に確率分布表を作成し，発生した$[0,1)$擬乱数を逆関数表で変換することで，遅延を計算している．GINEでは，指数分布，一様分布，正規分布，pareto分布，任意の表で指定した分布に従った乱数を発生できる．また，前後の相関も指定できる．

先頭フレームがQueueを出発するとき，次のフレームの送信予定時刻は，(3.2)式によって取得される．

$$\begin{aligned} \text{次フレーム送信予定時刻} &= \max(\text{次フレーム出発時刻}, \\ &\quad \text{先頭フレーム送信時刻} + \text{先頭フレームサイズ (bit)/帯域幅 (bps)}) \end{aligned} \tag{3.2}$$

“先頭フレームサイズ/帯域幅” でそのフレームの送信要する時間を算出し送信時刻を加算する．その時刻をこのフレームの次にリンクしているフレームの出発予定時刻と比較することで，次のフレームの実際の送信時刻を決定する．またこのQueueには，FIFO(First-in First-out)機能によって先に到着しているフレームを追い越さないように抑制する機能が追加されている．

フレーム損失は現在時刻において一様乱数を発生させ，フレームロスまたは一定のビットロス確率に従って生成される．GINEではランダムビットロス，フレーム(パケット)ロス(ランダム，マルコフ過程，パターン)の使用が可能である．

またQueue長が短くトラフィック量がリンク容量に比べて多い場合，Queueのオーバフローが発生することが多い．GINEではQueueのバッファサイズを指定できるため，オーバフローは起こりにくい設計になっている．故意にバッファサイズを小さくすると，Queueオーバーフローによる呼損を模倣できる．

3.3. 仮想ネットワークスタックの管理と通信方法

NETNSの生成時，ループバックを含むネットワークインターフェイスをOSや他のNETNSと共有できない．そこで，他のネットワークスタックとの通信のために特殊な仮想ネットワークデバイスであるVirtual Ethernet Pair (以下，veth)を用いる．このデバイスは対になっており，片方のデバイスでパケットを受信するともう片方のデバイスに転送する．各デバイスをそれぞれNETNSに渡すことで，NETNS間で相互通信可能となる．

vethのペアをNETNSにそれぞれ渡したときは，各vethにIPアドレスを付与することでNETNS間での通信が可能になる．しかし図5のように独自Queueを用いる場合はveth01，veth11はNETNSに渡してIPアドレスを付与しているが，GINEを起動しているホスト側にあるveth00，veth10はIPアドレスを付与してはいけない．付与してしまうとホストの外部あるいはホスト自体からのパケットがインターフェースへ直接転送されてしまう．また

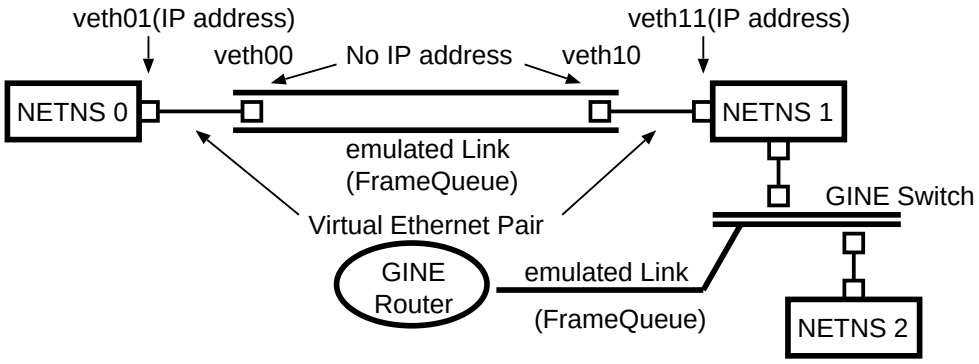

図 5: NETNS によるホスト/ルータ間での通信方法

GINE Switch を用いることで，複数の NETNS 間や GINE Router などとも通信することが可能になる．またこの Switch でも障害を発生させることができる．

3.4. GINE の改良点

本節では本論文で提案した GINE の改良点について説明する．

仮想ネットワークスタックの自動生成と端末を用いた管理

仮想ネットワークスタック (NETNS) を用いるとき，NETNS ごとにネットワークインタフェース設定等のコマンド実行が必要となる．しかし，仮想ネットワーク数が大きいとき，同数の端末を起動させると無駄なメモリ消費量が増え，エンドユーザの操作も困難になる．そこで，NETNS を GINE 内で管理しバックグラウンドで処理する機能が必要である．これは，fork 関数を用いてプロセスを分岐させ，擬似端末 (/dev/ptmx) で親プロセスと子プロセス (NETNS) をつなぎ，親プロセスから子にコマンド文字列を渡して実行することで実現した．

しかし，一部の NETNS に対しては，対話型操作も必要なので, 必要時に端末を開いて閉じる機能も実現した．端末には xterm を使用し，作成済の擬似端末のスレーブ側を接続する．

オブジェクトの保存/編集/読出機能の追加

さまざまなアプリケーションには任意のクラスのデータをファイルに保存したり，ファイルから復元したりする機能がある．特に動的に変化するモデルや複数の型を持つオブジェクトなどを生成するためには，オブジェクトの状態等を保存し復元する (オブジェクトの永続化，Persistent) 機能が必要である．しかし現在の GINE には，プログラム起動時にオブジェクトを生成するが，オブジェクトの保存機能は存在しない．

そこで，本論文では，GNU common C++の Persistent 機能 (Engine クラス) を用いて，オブジェクトの保存/読出機能を追加した．この機能を用いれば，保存/読み出しすべき情報は，各クラスにおいて指定する必要があるが，参照したオブジェクトも重複なしに処理できる．GINE で生成したオブジェクトはほとんどが双方向リスト形式で表現されているため，先頭オブジェクトのみを指定すれば末尾オブジェクトまで保存することができる．

GUI 機能の追加

現行 GINE の一般的な使用方法は，ネットワークトポロジごとに C++で，用意したライブラリクラスをつかったメインプログラムを記述し，それをコンパイルしてコマンド実行す

ることである．しかしこの方法では，模倣されたネットワークトポロジがソースプログラムを読まないと確認できない．また，プログラミングできない人にはエミュレータの利用が困難である．プログラム実行中のトポロジ修正，保存，読出しができない，等の欠点がある．

そこで本研究では，GUI 作成ツールを用いて GINE のライブラリを操作する GUI ツールを作成し，視覚的な操作を容易に実行できるようにする．例えば，NCTUns のように NETNS や FrameQueue のようなボタンを配置し，そのボタンを描画フィールドにドラッグ&ドロップしていくことでネットワークを構築し，さらに配置したオブジェクトを選択したときに別ウィンドウでネットワークや遅延，損失などを設定する．構築完了後，スタートボタンをクリックしたときに通信を開始する．さらにポーズボタンやストップボタンを用意し，時間経過に沿ったシナリオを再現できるようにする．最終的にはオブジェクト保存機能を用いてネットワークトポロジを保存でき，GUI アプリケーションから容易に操作できるようにする．

4. システムの評価

この節では，実験でエミュレータの動作が設定値に近いか，また，処理性能の限界を評価する．

使用したホストは，CPU Intel Xeon(Quad Core) 1.86GHz，2GB Memory，4GB swap，64bit OS である．ネットワークインタフェースは PCI-Express の 1000BASE-T である．

4.1. 独自 Queue によるフレーム遅延，損失測定

GINE を評価するために指数乱数を用いた片道一定遅延による往復遅延時間 (RTT) とフレーム損失を測定する．測定には ping(8 オクテットデータ送信) による ICMP 応答確認を用いた．測定結果を表 1 に示す．

表 1: 片道一定遅延による RTT 測定

Set(ms) theoretical	ping RTT(ms)			
	min	ave	max	mdev
direct	0.187	0.204	0.266	0.011
0.000	0.207	0.278	0.355	0.042
0.005	0.258	0.329	0.425	0.039
0.100	0.307	0.378	0.462	0.043
0.500	0.712	0.778	0.854	0.041
1.000	1.204	1.276	1.362	0.039
5.000	5.221	5.283	5.385	0.054
10.000	10.203	10.280	10.352	0.093
50.000	50.198	50.274	50.343	0.154
100.000	100.179	100.250	100.386	0.313
500.000	500.125	500.206	501.866	0.283
1000.000	1000.050	1000.130	1000.209	1.096

(`ping -i 0.1 -c 200 -s 8 TARGET_IPADDR`)

この指数乱数は， GNU libc のライブラリ drand48_r 関数を使用して生成した一様擬乱数を逆関数表で変換したものである．乱数発生関数は複数スレッドから呼ばれるのでリエントラントなものを使用する必要がある．実行速度を上げるために，逆関数の計算を乱数発生毎に実行するかわりに事前に逆関数表を生成しておく．また，実測で推定した任意の確率分布

表を与えれば，その分布に従う乱数を発生することができる．図 6 に GINE で模倣した指数分布による遅延密度を示す．

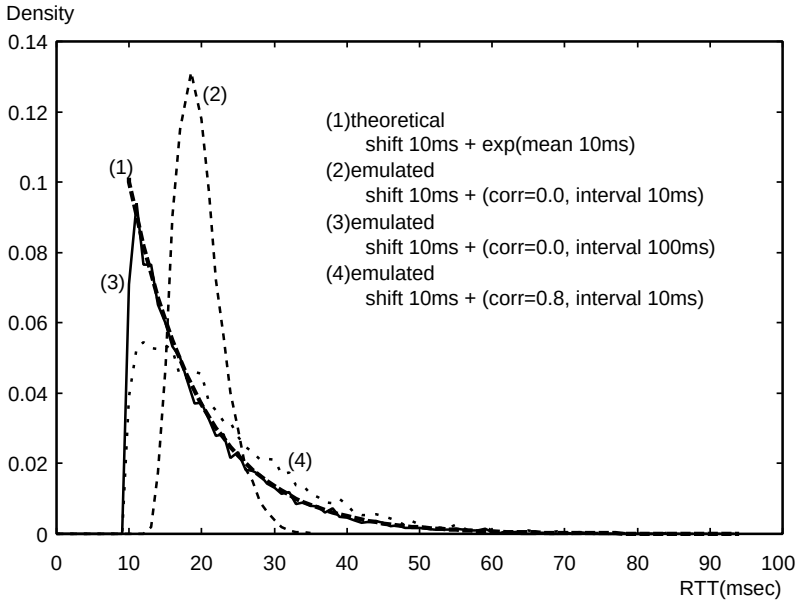

図 6: 指数分布による遅延の密度

(1) 10 マイクロ秒シフトと指数分布平均 10 マイクロ秒の和による密度関数の理論値
(2) GINE による 10 マイクロ秒のシフトと相関なし 10 マイクロ秒間隔での計算値
(3) GINE による 10 マイクロ秒のシフトと相関なし 100 マイクロ秒間隔での計算値
(4) GINE による 10 マイクロ秒のシフトと相関 0.8 の 10 マイクロ秒間隔での計算値

(3) は，(1) の理論値に十分近い．(4) では前後相関 0.8(正の値は前のフレーム遅延が大きければ次も大きい) の場合で，正の相関により分布がなだらかになったことが確認できる．

次にフレームの損失について議論する．1000 個のフレームを送信し， 5%と 20%の損失率を与えたときの確率を示す．損失は独立かつ一様分布に従う擬乱数で決定した．また試行を 10 回行ったときの最大値と最小値も求めた．表 2 より，期待値に非常に近い値を得ることができた．

表 2: フレーム損失率

Set theoretical	loss packet min	loss packet max	loss rate ave
5 %	48/1000	54/1000	5.2 %
20 %	196/1000	202/1000	19.8 %

(`ping -c 1000 TARGET_IPADDR`)

またビット損失率を 4%に設定したときの結果を表 3 に示す．ping で 10000，5000，2500bits を指定し，それぞれの期待値と実際に測定した損失率を比較した．

結果，それぞれのパケットサイズで期待値に近いパケット損失が発生することが確認された．パケットサイズを 1/2，1/4 にしたとき，損失率も比例して 1/2，1/4 となることも確認された．

表 3: ビット損失率

packet size(bits)	expectation	loss rate
10000	4 %	3.973 - 4.056
5000	2 %	1.984 - 4.132
2500	1 %	1.000 - 1.026

(`ping -s PACKETSIZE -c 10000 TARGET_IPADDR`)

4.2. フレーム転送性能評価

スループット測定には iperf[6] を，通信障害の測定には ping コマンドを用いる．

NFQUEUE 入出力方法を用いたフレーム転送性能

ネットワークエミュレータ内で GINE Router を 1 台から 60 台までのそれぞれの台数でパケット転送性能を評価する．GINE Router とは，仮想ネットワークスタックを用いた仮想ホストではなく，静的ルーティングのみの単純な機能を持つエミュレータ内のプログラムである．

図 2 のように Host A，Host B 間での TCP，UDP の最大スループットを測定した．また各リンクに遅延，損失などの障害などは設定しない．ホスト側の各ネットワークインタフェースは 1000BASE-TX(Broadcom BCM5754) を使用する．それに合わせて Queue のバンド幅も 1000Mbps に設定している．また独自 Queue 10 個ずつで 1 つの監視 Queue に登録した．10 回の試行でその平均値を求めた．

図 7 より，NFQUEUE の UDP は Divert Socket に比べて約 35 台まで高いスループットを測定できたが，その後は Divert Socket の測定値を下回る結果となった．これは NFQUEUE のプログラムの入出力処理の向上により改善されると考えられる．また全ての Queue に対して 1 つの監視スレッドで監視させているので，Queue の監視スレッドを増やし，分担させることでさらに向上すると考えられる．通信速度を重視した性能評価には Divert Socket を，容易に GINE を使用するためにはカーネル修正の必要がない NFQUEUE を使用することが妥当であると考える．

同様に Pentium M 1.20GHz 1GB(DDR2 SDRAM) 32bit OS でも測定した．すべての独自 Queue を 1 つの監視 Queue で監視させた場合，GINE Router 60 個で TCP スループット 6MB であった．このエミュレータは，短い待ち時間タイマを利用したマルチスレッドプログラムなので，少なくとも Dual core CPU を用いないと十分な性能が得られないと推定できる．

仮想ネットワークスタックを用いたフレーム転送性能

図 8 のように NETNS 上で動作した仮想ルータ，仮想ホストを配置し，IPv4 ネットワークを模倣する．

1. 通信性能 (NETNS0 - NETNS6 間，独自 Queue なし)
   TCP: 1255 Mbps，UDP: 980 Mbps と NETNS 間での通信では，高スループットを測定することができた．
2. 通信性能 (独自 Queue 使用，通信障害なし)
   各 NETNS 間に独自 Queue を配置した．表 4 より，1 ホップ数 (NETNS0 - NETNS 1) では最大 646Mbps，UDP スループットは 477Mbps を測定した．しかし独自 Queue を複数用いたことで，ホップ数に応じてスループットは徐々に減少する．これはホスト

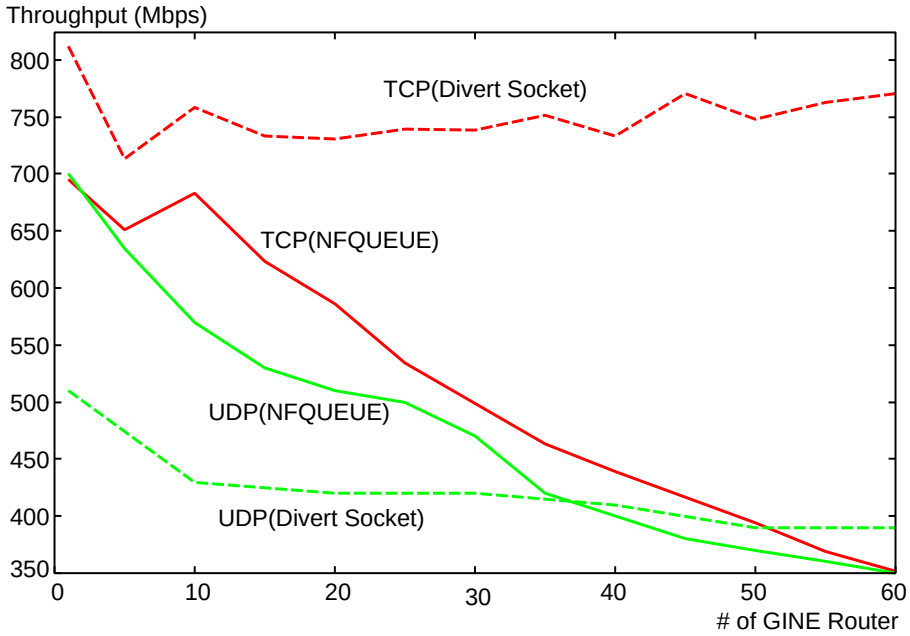

図 7: TCP，UDP スループット測定

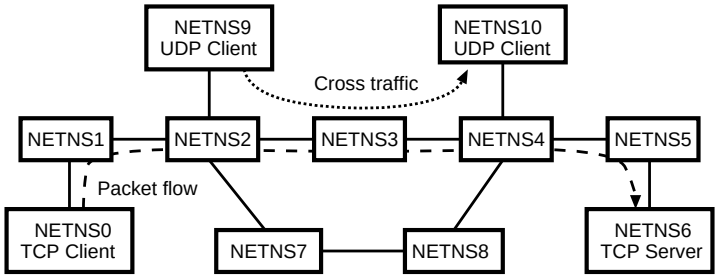

図 8: 小規模ネットワーク

自体の処理性能の低下によるものと考えられるが，プログラムや処理方法で改善できる可能性はある．

3. 通信性能 (独自 Queue 使用，遅延発生)
   表 5 のように各 NETNS1 - NETNS4 間でそれぞれ違うパラメータの遅延を発生させ，GINE の遅延機能に関して精度を測定した．結果，理論値である 600ms と非常に近い値を測定することができた．
4. 通信性能 (独自 Queue 使用，損失発生)
   表 6 のように各 NETNS1 - NETNS4 間でそれぞれ違うパラメータのロス率を設定し，GINE のフレームロス機能に関して精度を測定した．各リンクで 10%ごと損失を発生させているため，このときの理論値は 0.271 となる．測定した結果，理論値と非常に近い値を測定することができた．
5. 通信性能 (独自 Queue 使用，帯域制限)
   GINE によって帯域を指定し，そのときの測定値を表 7 に示す．帯域制限を 1000Mbps にしたときの測定値は 102Mbps であった．これは，表 4 より，独自 Queue によるスループットの限界が 115Mbps であったことを考えると妥当な値である．その他の値で

表 4: 帯域幅測定 (独自 Queue 使用，通信障害なし)

from NETNS0	TCP tput(Mbps)	UDP tput(Mbps)
to NETNS1 (hop 1)	646	477
to NETNS2 (hop 2)	350	224
to NETNS3 (hop 3)	215	162
to NETNS4 (hop 4)	165	120
to NETNS5 (hop 5)	130	95
to NETNS6 (hop 6)	115	89

表 5: 帯域幅測定 (独自 Queue 使用，遅延発生)

Delay point	set delay(ms)	results(ms)
NETNS1 - NETNS2	100	***
NETNS2 - NETNS3	200	***
NETNS3 - NETNS4	300	***
total	600	600.599

は，理論値に近い値を計測することができた．

6. クロストラフィックによる通信障害の発生とその振舞い

   図 8 のように NETNS9 - NETNS10 間でクロストラフィックを流し，NETNS0 - NETNS6 の通信状況を tcpdump を用いて確認した．実験方法として，NETNS0 - NETNS6 間で 30Mbps に制限されたバンド幅で TCP パケットを 100 秒間流している状態で，30〜40 秒，60〜70 秒の間，それぞれ 10Mbps，20Mbps の UDP クロストラフィックを発生させた．結果，図 9 のように，30〜40 秒の間で 10Mbps，60〜70 秒の間で 20Mbps スループットが低下していることが確認できた．

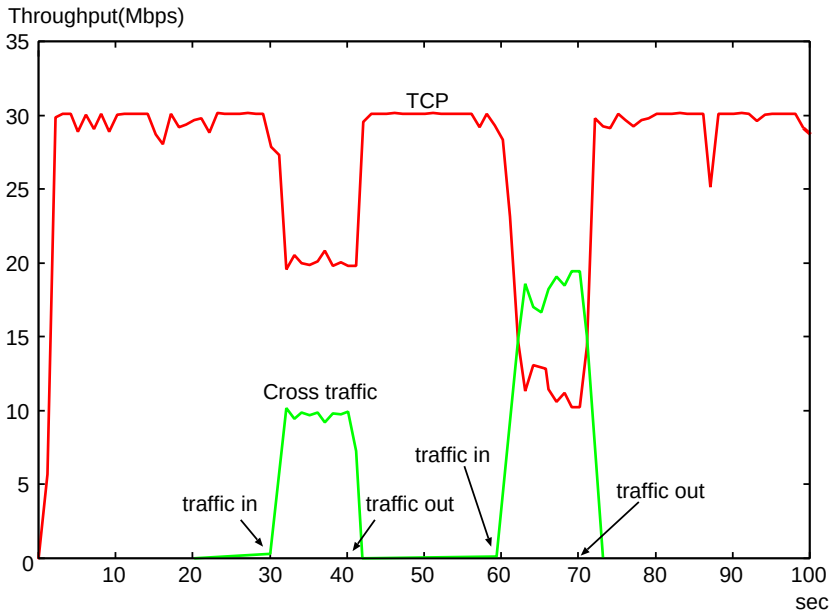

図 9: クロストラフィック測定結果

表 6: 帯域幅測定 (独自 Queue 使用，損失発生)

Loss point	set loss	results
NETNS1 - NETNS2	0.1 (10%)	***
NETNS2 - NETNS3	0.1 (10%)	***
NETNS3 - NETNS4	0.1 (10%)	***
total	$1 - 0.9^3 = 0.271$	0.264

表 7: 帯域幅測定 (独自 Queue 使用，帯域制限)

Limit point	set bw(Mbps)	TCP bw(Mbps)
NETNS0 - NETNS1	1000	102.03
NETNS0 - NETNS1	100	98.3
NETNS0 - NETNS1	20	19.10
NETNS0 - NETNS1	10	9.60
NETNS0 - NETNS1	1	98.4k
NETNS0 - NETNS1	0.1	9.92k

7. 仮想 Namespace 上で Virtual Web Server を起動させ，Web ページを閲覧できるか確認
   Web サーバは apache2 を起動することで実現している．各 NETNS 毎に apache2 を起動させる必要があるため，設定ファイルとプロセス番号を指定する必要がある．結果，NETNS0 から NETNS6 の Web ページが閲覧できることを確認することができた．
8. RIP デーモンの動作確認と，通信切断による別経路への遷移
   RIP デーモンを各 NETNS 上で動作させるためには，apache2 と同様に設定ファイルとプロセスをそれぞれ用意する必要がある．起動後，約 30 秒で経路情報を各ホストと交換し，通信することができる．NETNS3 でインターフェースをダウンさせたとき，NETNS0 から NETNS6 への経路は遮断されてしまう．このあと，RIP によって経路交換し，約 150〜170 秒で，NETNS6 へ再び通信することが確認できた．

本論文では IPv4 ネットワークモデルを使用したが，IPv6 ネットワークモデルも使用しネットワークを模倣できるように，GINE は構築されている．

最大ネットワークスタック数，スレッド数，ノード数

32 ビット OS では搭載メモリ量の上限は 4GB である．一方，64 ビット OS では理論上，16EB まで搭載可能である．また 32 ビット OS に比べて処理能力が向上し，短時間に大量のデータを処理することができる．

今回は検証ホストに 64 ビット OS を用い，2GB のメモリを搭載した．このホストの場合，単体で仮想ネットワークスタックを起動させたとき，約 1200 個起動したところで実メモリを使いきった．その後スワップ領域へ移行したが，急激に動作が遅くなった．さらに仮想ネットワークスタックを動作させ続けたとき，約 2000 個前後でホスト自体が完全に停止した．結果的に，約 1200 個前後であれば仮想ネットワークスタックを安定的に動作させることが可能であることがわかった．さらに搭載する実メモリを増やすことができれば，さらに起動数が増すと考える．また GINE Router に関しても，CPU やメモリ量に応じて起動数や，スループットが変化していく．

4.3. プログラムの実現

例として，図 8 のネットワークを表現するためのメインプログラムを図 10 に示す.

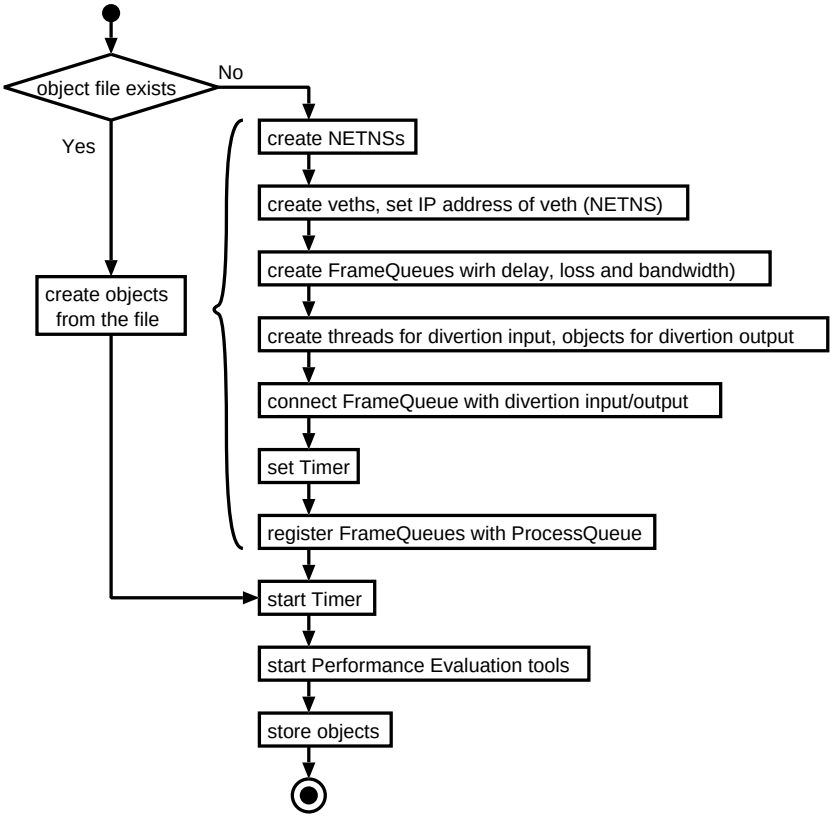

図 10: GINE メインプログラム例

この基本プログラムに加え，GINE Switch や GINE Router 等を生成し，独自 Queue と接続することで，大規模なネットワークを構築できる．またオブジェクト保存機能を追加することで，以前，構築したネットワークを再度読出し，同じ挙動を再現できるようになる．

5. おわりに

本研究では，ネットワークエミュレータの機能を拡張し，その性能を評価した．その結果，仮想ネットワークスタックを利用し，1 台の PC で数 10 から数 100 のホストから構成されるネットワークを模倣し，動的ルーティングと仮想ホストでのサーバ起動等が実用的であることがわかった．また，Linux カーネル標準の横取り機能を用いて，数 100Mbps 以上の高スループットを実現できた.

3.4 節で述べた改良点のうち，オブジェクトの保存と読み出しは完成した．GUI は，drag and drop によるオブジェクトの生成，操作パネル等基本的要素の作成に留まっていて，完成が今後の課題である.

参考文献

[1] Ihara, A., Murase, S., Goto, K.: *IPv4/v6 network emulator using divert socket*, Proc. of 18th International Conference on Systems Engineering(ICSE2006，Coventry, UK), pages 159–166, September 2006.

[2] Free Software Foundation: *Gnu common C++*, http://www.gnu.org/software/commoncpp/ (accessed Apr. 2009).

[3] Harald, W.: *libnetfilter_queue project* http://www.netfilter.org/projects/ (accessed Apr. 2009).

[4] Network Namespace: *Linux Containers*, http://lxc.sourceforge.net/network.php (accessed Apr. 2009).

[5] NIST Net: *NISE NET Home Page*, http://snad.ncsl.nist.gov/itg/nistnet/ (accessed Apr. 2009).

[6] NLANR. Iperf: *The tcp/udp bandwidth measurement tool*, http://dast.nlanr.net/projects/Iperf/ (accessed Apr. 2009).

[7] Wang, S., and Chou, C.: The Design and Implementation of the NCTUns Network Simulation Engine, *Elsevier Simulation Modelling Practice and Theory*, Amsterdam, Nederlanden, pp. pp.57 – 81 (2007), http://nsl10.csie.nctu.edu.tw/support/documentation/simpat2007.pdf (accessed Apr. 2009).

[8] Sugiyama, Y. and Goto, K.: *Design and Implementation of a Network Emulator using Virtual Network Stack*, Proc. of International Symposium on Operations Research and Its Applications(ISORA'08), pp.351-358, November 2008.

[9] Zec, M., and Mikuc, M: *Operating System Support for Integrated Network Emulation in IMUNES*, Operating System and Architectural Support for the on demand IT InfraStructure / ASPLOS-XI, Boston,America (2004), http://tel.fer.hr/zec/papers/zec-mikuc-04.pdf (accessed Apr. 2009).