リンクエミュレータの過渡解析

— dummynet 制御における遅延 —

三輪 信介* 知念 賢一†

ネットワークリンクの遅延や帯域，パケットロス率などの特性をエミュレートする機構としてリンクエミュレータがある．リンクエミュレータの各特性の制御の精度や特性の設定を行ってから制御が安定するまでの期間などの過渡特性は，リンクエミュレータを利用してネットワークリンクの特性を動的制御に影響を及ぼす．そこで，本稿では，リンクエミュレータとして dummynet を取り上げ，その過渡特性の解析について述べる．

Analyzing Transitive Behaviours on Link-Emulators

— Delay Control on dummynet —

Shinsuke Miwa‡ Kenichi Chinen§

Link-emulators are capable of emulating attributes of a network link such as delay, bandwidth and packets loss rate. When parameters of a link-emulator were dynamically changed, accuracy of controlling attributes and timescale of stabilizing attributes will be affected by transitive behaviours on the link-emulator. In this paper, we will describe a analyzing transitive behaviours of dummynet as a link-emulator.

1 はじめに

ネットワークに関する実験を行う上で，リンクの遅延や帯域，パケット損失率などを実験の内容に合わせて変化させる必要が生じる．このような場合に，実際に必要な特性を有する物理リンクを用意するのは困難もしくは高コストであるため，リンクの特性をエミュレートするリンクエミュレータが広く用いられている．

ここでは，特定のリンク特性を模倣することをリンクエミュレーション，それを実現する機器やソフトウェアを包括してリンクエミュレータ，特にソフトウェアの場合は，リンクエミュレーションソフトウェアと呼ぶこととする．

我々は，これまで実機ベースのネットワーク実験向け汎用実験設備である StarBED およびその実験支援ソフトウェア SpringOS[1] や仮想 PC によるセキュリティ向けの実証環境である VM Nebula[2, 3] などを研究開発してきた．これらにおいても，実験に応じたリンク特性の変更が求められており，現在開発中の次期実験支援ソフトウェアでは，リンクエミュレータによるリンク特性の制御機能の付加を予定している．

そこで，本稿では，既存のリンクエミュレーションソフトウェアの制御の精度や設定変更時の過渡特性などの計測および解析について述べる．具体的には，FreeBSD の dummynet[4] を取り上げ，SmartBits[5] を用いた計測の結果とその解析について述べる．

この解析により，設定変更から実際に設定が反映されるまでの期間（設定移行期間）を明らかにし，リンクエミュレータの制御の仕方など，次期実験支援

*情報通信研究機構 情報通信セキュリティ研究センター
†北陸先端科学技術大学院大学 情報科学研究科
‡Traceable Secure Network Group, Information Security Research Center, National Institute of Information and Communications Technology
§School of Information Science, Japan Advanced Institute of Science and Technology

表 1: 計測対象ノード F1 の構成

項目	構成
CPU	Pentium4
Memory	2GB
NIC	1000Base-T×6
OS	FreeBSD 5.4 Release

ソフトウェアの設計に反映させる予定である.

2 計測環境

まず, 計測に用いた OS やソフトウェア, 計測機器, その他の環境などの構成について述べる.

2.1 dummynet

dummynet は FreeBSD に標準で付属しているトラフィックシェイパーで, 帯域, 遅延, パケット損失率を制御することができる. kernel のオプションモジュールとして提供されており, IP ファイアウォールを提供する ipfw モジュールのインタフェイスである ipfw コマンドをユーザインタフェイスとして利用する. 帯域制御には, WFQ2+ (Worstcase Fair Weighted Fair Queueing; 最低限保証重み付き均等保証キュー) を用いている.

2.2 計測対象ノード・計測機器

計測は, StarBED2 上で米 SPIRENT 社の SmartBits を用いて行った.

計測対象のノードとして, StarBED 上のグループ F のノード 1 台 (以降, ノード F1) を用いた. ノード F1 の構成を表1 に示す. ノード F1 のカーネルに dummynet を用いるための下記の設定を追加して再構築し, 利用した.

```
1:  options IPFIREWALL
2:  options IPFIREWALL_VERBOSE
3:  options IPFIREWALL_DEFAULT_TO_ACCEPT
4:  options HZ=1000
5:  options DUMMYNET
```

1 行目は dummynet を利用する際のベースとなる kernel モジュールである IP ファイアウォールのモジュールを有効にする. 2 行目では IP ファイアウォールのログ出力等を有効にする. 3 行目は通常の IP ファイアウォールが初期状態はルールにマッチしないパケットはすべて deny (拒否) であるのに対し, accept (許可) に変更する. IP 以外のパケットも含めてすべて dummynet に流すために必須である. 4 行目は kernel のタイマーの粒度を決める. dummynet を用いる際の推奨値が 1000 であり, 今回はこの推奨値に設定している. 5 行目は dummynet の kernel モジュールを有効にする.

表 2: 計測に利用した SmartBits の構成

項目	構成
本体	SmartBits 600B
モジュール	LAN-3310A
	GBIC 1000Base-T

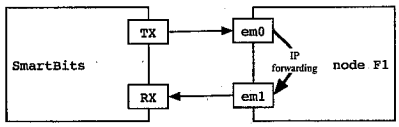

図 1: 計測環境の構成

SmartBits は, ネットワークの性能を計測する測定器で, 送信ポートから発信され, 受信ポートまで届いた通信の内容から, 下記のような対象ネットワークのさまざまな性能を計測できる.

- パケットが発信されてから受信されるまでの遅延を時間で測定
- 発信パケット数と受信パケット数の比からパケット損失率を測定
- 送受信可能な最大量から帯域を測定

今回 SmartBits は, 表2 に示す構成で, 1000Base-T に対応している計測モジュールを導入して用いた.

2.3 計測環境の構成

計測は, 図1 のような構成で行った. SmartBits の送信ポートからノード F1 のネットワークインタフェイス em0 に送信されたパケットは, IP Forwarding され, ネットワークインタフェイス em1 から SmartBits の受信ポートに転送される. 今回は IP Forwarding を利用しているが, dummynet はブリッジでの利用も可能である.

ノードF1上での外乱を軽減するために，いくつかのプロセス（init，login，ttyとカーネルプロセス）を残して，他は停止して実験を行った．

3 遅延の測定

遅延の計測は，SmartBitsの“Latency overtime”テストで実施した．“Latency overtime”テストは，指定した時間間隔ごとに受信したデータフレームの数および，遅延の最小値と最大値もしくは平均値を計測する．

今回は，計測間隔は1msに設定し，1ms毎の受信フレーム数および遅延の平均値を計測した．フレームサイズは既定値の124バイトとした．

3.1 予備計測

まずは，SmartBitsのインタフェイス同士を直結して遅延を計測し，計測機器自身の遅延を把握した．その結果，計測機器の遅延は0.7μsであり，$\pm$1μs程度の計測粒度を持つことが確認できた．dummynetの遅延の設定粒度がmsであることから，計測機器の計測粒度には全く問題がない．

続いて，dummynetを用いない場合の遅延，すなわちノードF1のFreeBSDでのIP forwardingのみの遅延性能を計測した．SmartBitsの計測パラメータの一つである計測ネットワークの利用率（最大利用帯域の実効帯域に対する割合で要求値）の設定を1%から100%まで変更し，計測した場合でも，遅延は最大594.2μsであり，さらに今回対象とした利用率10%では最大155.2μsであるため，$\pm$1ms程度の精度での解析には問題がないことを確認した．

さらに，dummynetを用いた場合に正しく計測できるかを確認した．遅延を0msに設定した場合には問題はなかったが，遅延を100msに設定した場合，計測ネットワークの利用率の設定値が25.1%より上になると正しく計測できなかった．また，ネットワークの利用率の設定値が15.8%以上の場合，dummynetの遅延設定を変更しようとしてもipfwコマンドが終了しないなどの問題が発生した．

原因が特定できておらず，解決方法も発見できていないため，今回は，計測ネットワークの利用率の設定値を10%に固定し，計測することにした．GbEの10%としても帯域が100Mbpsまでのネットワークに対する遅延性能を計測できるため，リンクエミュレータによる制御の対象をFastEthrenet程度までと考えるのであれば，問題ないと判断した．

3.2 過渡計測

dummynetの遅延制御の性能および過渡特性を見るために，遅延の設定値を計測中に変更し，設定移行期間や設定変更の前後で遅延がどのように変化するかを計測した．具体的には，遅延の設定値を0msから100msもしくは100msから0msに25ms刻みに増減させた．変更は1秒ごとに行い、1msあたりの遅延の平均値を計測した．

計測には下記の簡単なスクリプトを用い，SmartBitsによる計測の開始・停止とスクリプトの起動は手動で行った．

```
1: ipfw -f flush
2: ipfw pipe 10 delete
3: ipfw <管理用の通信を許可>
4: ipfw add 100 pipe 10 <em1から発信される通信>
5: ipfw pipe config delay <設定する遅延> ※
6: sleep 1

※遅延を変更して繰り返し
```

1行目のipfw flushは，ipfwが保持するルールセットを初期化する．2行目のipfw pipe 10 deleteは今回dummynetに用いるパイプを削除する．3行目は，dummynetを通さない管理用の通信を通過させるルールセットを設定する．4行目は，今回dummynetを適用する通信をノードF1のem1から発信される通信に限定している．5行目では実際にdummynetに遅延を設定する．6行目では，次の遅延設定の変更まで1秒間，実行を停止する．

以降では，スクリプト中の5行目のipfw pipe config ...を「遅延設定の変更」，6行目のsleep 1を「1秒間の停止」と呼ぶこととする．

3.3 結果の予想

増加，減少いずれの場合においても，遅延は毎秒25msずつ変更する．そのため，遅延を0msから100msに上げていく場合には，グラフは1秒ごとに遅延が25msずつ上がっていく階段状になり，段が上がる直前に遅延の差分を吸収するために25ms

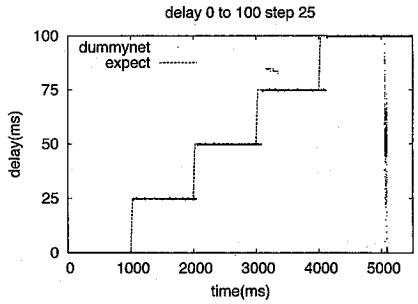

図 2: 遅延: 0ms から 100ms

の空白期間ができていると予想される．逆に，遅延を 100ms から 0ms に下げていく場合には，グラフは 1 秒 − 25ms すなわち 975ms ごとに遅延が 25ms ずつ下がっていく階段状になると予想される．また，遅延の差分を吸収するために，段が落ちる前後に何らかの変動があると予想される．さらに，最初 100ms は初期の遅延を発生するために空白期間になると予想される．

なお，今回はスクリプト上でタイマーとして `sleep` コマンドを用いているため，正確に 1000ms 毎に遅延設定が変更されているとは限らないことも予想される．

これらの予想から，各段の長さや，段が変わる前後の挙動がどうなっているかを留意点として，解析する必要がある．

4 過渡解析

計測結果について，結果の予想に照らし合わせて過渡解析を試みる．

なお，スクリプトの起動や各コマンドの起動が計測データ上のどの時点で行われたのかは，計測データには直接記録されない．そのため，データに顕著な変化が現れた点を遅延設定の変更の効果が現れた時点と扱うこととする．

計測は 100 回行い，結果はその統計値である．

4.1 遅延の増加

遅延設定を 0ms から 100ms まで増加させていった場合の計測結果のグラフを予想と重ねて図2 に示す．グラフの縦軸は遅延，横軸は時間である．

グラフを見ると，予想された結果より，若干各段の幅が広く，右にずれていく様子がうかがえる．[1]

データを見ると一段目が平均 1030ms，二段目以降は平均 1007ms となっている．一段目が長い理由が初期化部分が除去しきれずに残ったため，もしくは遅延の設定値が 0 の場合の処理が違うためと仮定すると，遅延を 0 以外に設定した場合の各段の長さは約 1007ms と考えられる．また，段が変わる直前には，設定遅延の増加分から予想された 25ms に近い平均 24ms の空白期間が存在し，それ以外には段の切れ目で特に目立った変動はない．各段の遅延も設定遅延周辺 ±1ms に沿っている．

各段の長さは，3.2 節のスクリプトの「1 秒間の停止」と繰り返し後の「遅延設定の変更」の所要時間合計なので，それぞれの所要時間を $t_{\texttt{sleep}}, t_{\texttt{ipfw}}$ とした場合，`sleep 1` を正確に 1000ms であると仮定し，`ipfw` コマンドが効力を現すまでに必要な時間を α_{up} とすると，

$$t_{\texttt{sleep}} + t_{\texttt{ipfw}} = 1000\text{ms} + \alpha_{up} \fallingdotseq 1007\text{ms}$$
よって，$\alpha_{up} \fallingdotseq 7\text{ms}$

と考えることができる．

よって，シェルスクリプトを利用して遅延を増加させていく場合，遅延の増加分と同程度の空白期間を考慮すると，

「遅延の増加分 + 7ms」

の設定移行期間で設定した遅延に変更することができる．

4.2 遅延の減少

遅延設定を 100ms から 0ms まで減少させていった場合の計測結果のグラフを予想と重ねて図3 に示す．グラフの縦軸は遅延，横軸は時間である．

グラフを見ると，予想された結果より，

(a) 各段が若干長いこと

(b) 遅延の変更が数 ms の時間をかけて行われていること

(c) 段が変わった直後からしばらく遅延が若干大きくなっていること

[1] なお，5000ms を少し越えた部分に散逸する点は，100ms に設定された遅延から次の試行に移る際に 0ms に設定される際の移行部分と考えられる．

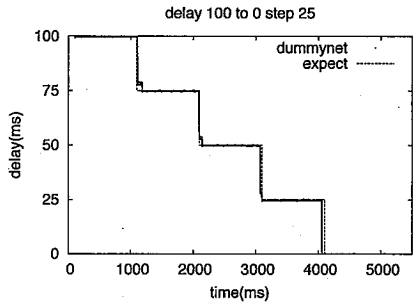

図 3: 遅延: 100ms から 0ms

が見て取れる.

(a) は, データを見ると一段目は平均 1106ms で二段目以降は平均 982ms となっている. 一段目に最初の設定遅延分 100ms が含まれていると仮定すると, 実際には 1006ms であると考えることができる. これは,「1 秒間の停止」と繰り返し後の「遅延設定の変更」の所要時間合計なので, それぞれの所要時間を $t_{\texttt{sleep}}$, $t_{\texttt{ipfw}}$ とした場合, `sleep 1` を正確に 1000ms であると仮定し, `ipfw` コマンドが効力を現すまでに必要な時間を α_{down1} とすると,

$$\begin{aligned} t_{\texttt{sleep1}} + t_{\texttt{ipfw}} &= 1000\text{ms} + \alpha_{down} \\ &= 1006\text{ms} \\ \text{よって, } \alpha_{down1} &= 6\text{ms} \end{aligned}$$

二段目以降は設定遅延の差分 25ms だけ送出時間が短いと考えられ, 予測値は 975ms である. 計測値は平均 982ms であり, コマンドが効力を現すまでに必要な時間が含まれていると考えられる. これに設定遅延の差分 25ms を足したものが「1 秒間の停止」と繰り返し後の「遅延設定の変更」の所要時間合計なので, それぞれの所要時間を $t_{\texttt{sleep}}$, $t_{\texttt{ipfw}}$ とした場合, `sleep 1` を正確に 1000ms であると仮定し, `ipfw` コマンドが効力を現すまでに必要な時間を α_{down2} とすると,

$$\begin{aligned} t_{\texttt{sleep1}} - 25\text{ms} + t_{\texttt{ipfw}} &= 975\text{ms} + \alpha_{down2} \\ &= 982\text{ms} \\ \text{よって, } \alpha_{down2} &= 7\text{ms} \end{aligned}$$

と考えられる.

(b) は, データを見ると変更開始から平均 4ms の間に, 設定した遅延に近づくように遅延が変更されていくのが見て取れる.

(c) の遅延が若干大きい期間は, 設定遅延が小さくなるにつれ短くなり, 設定遅延が 0ms としたときには見受けられない. 段差の高さは, 平均 2ms で, 期間は, それぞれ平均 74ms, 48ms, 26ms であり, 設定した遅延と相関があると思われる.

これらの結果から, シェルスクリプトを利用して遅延を減少させていく場合, 設定遅延以下に収束するまでには,

「設定遅延 + 6〜7ms + 4ms」

の設定移行期間を必要とすると考えられる.

5 今後の課題

計測手法の問題として, 今回は遅延の変更間隔を `sleep` コマンドを用いて実現したが, `sleep 1` が正確に 1000ms を保証しているわけではなく, 制御粒度も不明である. また, 遅延の変更はシェルスクリプト上で `ipfw` コマンドを用いて行っており, 正確にコマンド実行時点を特定できない. そこで, シェルスクリプトではなく, 専用の制御プログラムを用意し, `ipfw` のシステムインタフェイスやより詳細なタイマーである `usleep` を用いるなど, より正確な `dummynet` の制御が必要であると考える.

今回は, 25ms 間隔で遅延を増減した場合についてのみ計測したが, 遅延の設定間隔などを変更しながら試行し, データのより詳細な分布を確認する必要があると考えている.

また, 今回計測したのは遅延のみだったが, `dummynet` は帯域やパケット損失率, キューの長さなども変更可能であり, これらに関しても同様の解析が必要であると考える.

さらに, 今回は FreeBSD に標準の `dummynet` のみを対象としたが, 同様に Linux に標準の `netem`[6] や, Linux ベースで高機能な NIST Net[7] などは試していない. 今後これらについても同様の解析を行う予定である.

6 おわりに

本稿では, FreeBSD に標準で付属している `dummynet` の遅延制御について, StarBED2 上で SmartBits を用いて計測し, 過渡解析を行った.

解析によって, `dummynet` を利用してシェルスクリプトによって遅延を制御する際には,

- 遅延を増加させる場合,
  「遅延の増加分 + 7ms」

- 遅延を減少させる場合,
  「設定遅延 + 10ms〜11ms」

の設定移行期間が必要との結果が得られた.

この結果を基に, SpringOS や VM Nebula2 などの次期実験支援ソフトウェアにおけるリンクエミュレータの制御手法に反映させる予定である.

参考文献

[1] Toshiyuki Miyachi, Ken-ichi Chinen, Yoichi Shinoda, "Automatic Configuration and Execution of Internet Experiments on an Actual Node-based Testbed" Tridentcom2005, Trento, Italy, ISBN 0-7695-2219-X, pp.274–282, Feb. 2005.

[2] 三輪 信介, 滝澤 修, 大野 浩之, "仮想 PC インターネットセキュリティ実験環境『VM Nebula』の設計と構築", 電子情報通信学会, 2003 年 暗号と情報セキュリティシンポジウム (SCIS2003), Jan. 2003.

[3] 三輪 信介, 大野 浩之, "インターネットセキュリティ実験環境『VM Nebula2』の設計と構築", 電子情報通信学会, 2006 年 暗号と情報セキュリティシンポジウム (SCIS2006), Jan. 2006.

[4] L. Rizzo, "Dummynet: a simple approach to the evaluation of network protocols", *ACM Comupter Communication Review*, Jan. 1997.

[5] Spirent SmartBits, Trusted Industry Standard for Router and Switch Testing, ⟨ URL: `http://www.spirentcom.com/analysis/technology.cfm?media=7&ws=325&ss=110`⟩.

[6] S. Hemminger, "Network Emulation with NetEm", linux.conf.au 2005, Apr. 2005.

[7] M. Carson and D. Santay, "NIST Net — A Linux-based Network Emulation Tool", *ACM Computer Communication Review*, June. 2003.