DUMMYNET 拡張キット Ver. 1.0 説明書

株式会社 インターネットオートモビリティ研究所

■ DUMMYNET 拡張キットの概要

DUMMYNET 拡張キットは FreeBSD の標準機能である DUMMYNET の機能拡張を行うためのカーネルパッチおよび API ライブラリのセットです。本拡張キットを用いていただくことにより、DUMMYNET 本来の機能である伝送遅延の挿入や伝送エラー率の設定をより柔軟におこなうことが可能となります。また、本拡張キットは、容易にお客様が開発されるアプリケーションプログラムよりこれらの設定が行えるような API ライブラリがセットとなっています。

DUMMYENT 拡張キットでは下記のようなモジュールを提供しています。

◆ DUMMYNET 拡張機能

- DUMMYNET では伝送遅延や伝送エラー率を固定値として設定することができますが、本拡張機能を用いることにより、様々な分布関数に基づいた伝送遅延や伝送エラー率の揺らぎを挿入することが可能となります。確率分布の配列として与えますので、あらゆる分布関数に対応可能です。

◆ API ライブラリ

- 一様分布、正規分布、対数正規分布を使った伝送遅延、伝送エラー率を設定するための API を提供しています。
- ブリッジ設定を行うための API を提供してます。この API を用いることにより、より簡単にブリッジ設定を行うことが可能となります。
- IPFW のパイプを作成/削除するための API を提供しています。この API を利用することにより、通常、ipfw コマンドを用いて設定するパイプの作成/削除をアプリケーションプログラムから直接おこなうことが可能となります。
- API は C 言語の関数として提供しています。

また、DUMMYNET 拡張キットは下記の環境で動作するように設計されています。

- OS　　FreeBSD 5.1-RELEASE
- CPU　　Pentium III 1.0GHz 以上
- メモリ　　256MB 以上
- LAN　　10Base-T 以上

■ DUMMYNET 拡張キットのインストール

本章では DUMMYNET 拡張キットのインストール方法について説明します。DUMMYNET 拡張機能のインストールの大まかな流れは、1)カーネルおよび関連コマンドの再構築、2)API ライブラリのインストールです。

□ カーネルおよび関連コマンドの再構築

まず、カーネルの再構築を行います。展開した DUMMYNET 拡張キットにあるパッチを FreeBSD のカーネルソースにあててください。以下の例ではディレクトリ EDN に DUMMYNET 拡張キットを展開してあると仮定しています。

```
# cd /sys
# patch –p1 < EDN/sys.diff
```

この作業を行うことにより、i386/conf ディレクトリの下に EXTDUMMYNET というカーネルコンフィグレーションファイルができますので、これを使ってカーネルを再構築します。

```
# cd /sys/i386/conf
# config EXTDUMMYNET
# cd ../compile/EXTDUMMYNET
# make depend
# make
```

カーネルの構築が終了したらカーネルをインストールしてリブートしてください。

```
# make install
# reboot
```

これでカーネルのインストールは終了ですが、このままでは ipfw コマンドが利用できません。これは、DUMMYNET 拡張キットがいくつかのヘッダファイルを変更しているためです。そこで、ヘッダファイルを入れ替え、ipfw コマンドを再コンパイルします。まずはヘッダファイルの入れ替えです。

```
# cd /usr/include/netinet
# mv in.h in.h.org
# mv ip_dummynet.h ip_dummynet.h.org
# cp /sys/netinet/in.h /sys/netinet/ip_dummynet.h /sys/ip_extdummynet.h .
```

これでヘッダファイルの入れ替えが行われました。次に ipfw の再コンパイルです。

```
# cd /usr/src/sbin/ipfw
# make
# make install
```

これで、DUMMYNET 拡張キットのカーネルおよび関連コマンドに関するインストールが終わりました。

□ API ライブラリ

API ライブラリは特にインストールを必要としません。次章の API ライブラリの使い方を参考にしてお客様が開発されるアプリケーションとともにコンパイルしてお使いください。また、本 API ライブラリを御使用になる場合は math ライブラリが必要になります。コンパイル時(リンク時)に「-lm」をご指定ください。

コンパイル例:

```
% gcc –o test main.c libedn.c -lm
```

■ API ライブラリに関する説明

本章では、DUMMYNET 拡張キットが提供するアプリケーションプログラミングインターフェイス(API)について説明します。

□ EDN_bridge_cfg ― DUMMYNET のブリッジの設定を変更する

この関数では DUMMYNET の標準機能であるブリッジに関する設定を行います。設定はカーネル MIB である net.link.ether.bridge_cfg に設定するものと等価な関数です。

関数型

int EDN_bridge_cfg(char *cfg)

引数

cfg: net.link.ether.bridge_cfg に設定する値。

戻り値

0: 成功
-1: 失敗

□ EDN_add_pipe ― パイプを設定する

この関数はファイアウォール機能を用いて、パイプへのフィルタを設定します。

関数型

int EDN_add_pipe(int rule, int pipe, char *ifname)

引数

rule: ルール番号。
pipe: パイプ番号。
ifname: フィルタする受信インターフェイス名。受信インターフェイスを指定しないときは NULL を設定する。

戻り値

0:　　成功

-1:　　失敗

□ EDN_del_pipe ― パイプを削除する

この関数は EDN_add_pipe で作成したパイプへのフィルタを削除します。

関数型

int EDN_del_pipe(int rule)

引数

rule:　ルール番号。

戻り値

0:　　成功

-1:　　失敗

□ EDN_pipe ― パイプを設定する

この関数はパイプの作成、設定および削除を行います。設定できる値としては、バンド幅、伝送エラー率、伝送遅延があります。

関数型

int EDN_pipe(int num, int cmd,
　　double bw,
　　int etype, double e1, double e2,
　　int dtype, double d1, double d2)

引数

num:　パイプ番号。

cmd:　コマンド。設定可能なコマンドは下の通り。

　　EDN_PIPE_DELETE　　パイプを削除する。

　　EDN_PIPE_CONFIG　　パイプのパラメータを設定するただし、パイプが存在しない場合は作成する。

bw:　バンド幅を bps で指定する。

etype: エラー率のパターンを指定する。設定可能なパターンは下の通り。

EDN_PIPE_ETYPE_ERR: 一定値。

EDN_PIPE_ETYPE_UERR: 一様分布。

EDN_PIPE_ETYPE_NERR: 正規分布。

EDN_PIPE_ETYPE_LERR: 対数正規分布。

e1: 一定値の場合⇒エラー率。

一様分布の場合⇒エラー率の下限。

正規分布の場合⇒エラー率の平均。

対数正規分布の場合⇒m

e2: 一定値の場合⇒ダミー変数。

一様分布の場合⇒エラー率の上限。

正規分布の場合⇒エラー率の分散。

対数正規分布の場合⇒σ

dtype: 遅延のパターンを指定する。設定可能なパターンは下の通り。

EDN_PIPE_DTYPE_ERR: 一定値。

EDN_PIPE_DTYPE_UERR: 一様分布。

EDN_PIPE_DTYPE_NERR: 正規分布。

EDN_PIPE_DTYPE_LERR: 対数正規分布。

d1: 一定値の場合⇒遅延の値をミリ秒で指定。

一様分布の場合⇒遅延の下限をミリ秒で指定。

正規分布の場合⇒遅延の平均をミリ秒で指定。

対数正規分布の場合⇒m

d2: 一定値の場合⇒ダミー変数。

一様分布の場合⇒遅延の上限をミリ秒で指定。

正規分布の場合⇒遅延の分散をミリ秒で指定。

対数正規分布の場合⇒σ

戻り値

0: 成功

-1: 失敗

■ 技術情報

本章では DUMMYNET 拡張キットを使った設定が引き起こす効果について説明します。

□ 遅延として一定値を用いた場合の効果

遅延として一定値を用いた場合、遅延の揺らぎは殆どなく、常に設定した値が遅延として現れるはずです。例えば、本キットが提供する API を使って下のように設定した場合、遅延は常に 500 ミリ秒となります。

```
EDN_pipe(10, EDN_PIPE_CONFIG, 0.0,
        EDN_PIPE_ETYPE_ERR, 0.0, 0.0,
        EDN_PIPE_DTYPE_DELAY, 500.0, 0.0);
```

実際にブリッジ設定の片道に対し上記のような設定を行ない、0.1 秒ごとに 3000 回 ICMP Echo を送信したときの RTT の頻度分布を図 1 に示します。この図から遅延は常に 500 ミリ秒となっていることが読み取れます。

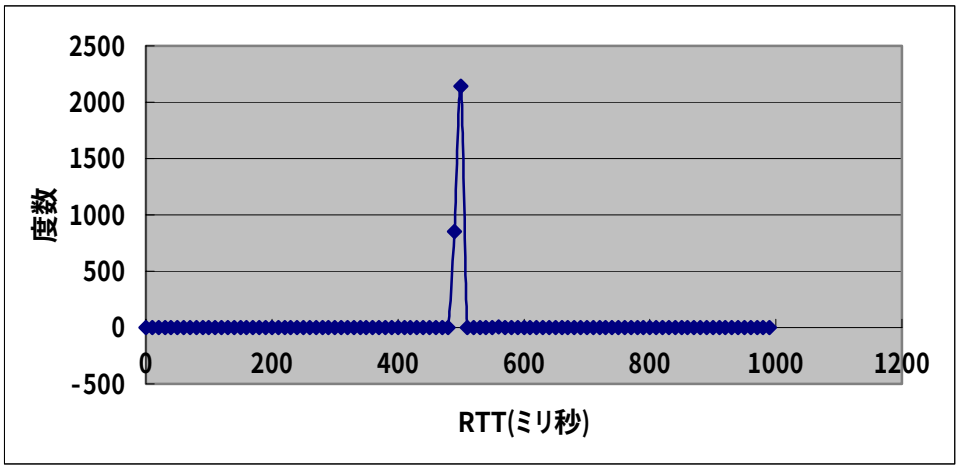

図 1: 遅延として一定値を用いた場合の遅延の頻度分布
(設定: 遅延 500msec)

□ 遅延として一様分布を用いた場合の効果

遅延として一様分布を用いた場合、遅延は設定範囲内で一様な揺らぎとして現れます。例えば、本キットが提供する API を使って下のように設定した場合、遅延は 400 ミリ秒から 600 ミリ秒の間で一様な揺らぎとなって現れます。

```
EDN_pipe(10, EDN_PIPE_CONFIG, 0.0,
        EDN_PIPE_ETYPE_ERR, 0.0, 0.0,
        EDN_PIPE_DTYPE_UDELAY, 400.0, 600.0);
```

実際にブリッジ設定の片道に対し上記のような設定を行ない、0.1 秒ごとに 3000 回 ICMP Echo を送信したときの RTT の頻度分布を図 2 に示します。この図から遅延は 400 ミリ秒から 500 ミリ秒の間で変化しているとことが読み取れます。しかし、分布が一様ではなく、遅延が大きくなるほど度数も大きくなっています。これは 0.1 秒ごとに ICMP Echo を送信しているため、遅延の揺らぎのために本来小さな遅延となったはずのパケットが前のパケットを追い越すことが出来ず、結果大きな遅延として送信されることに起因しています。この問題は送信間隔を十分にとることによって解決できます。送信間隔を 1 秒に設定した時の頻度分布を図 3 に示します。こちらの図では分布がほぼ一様になっています。

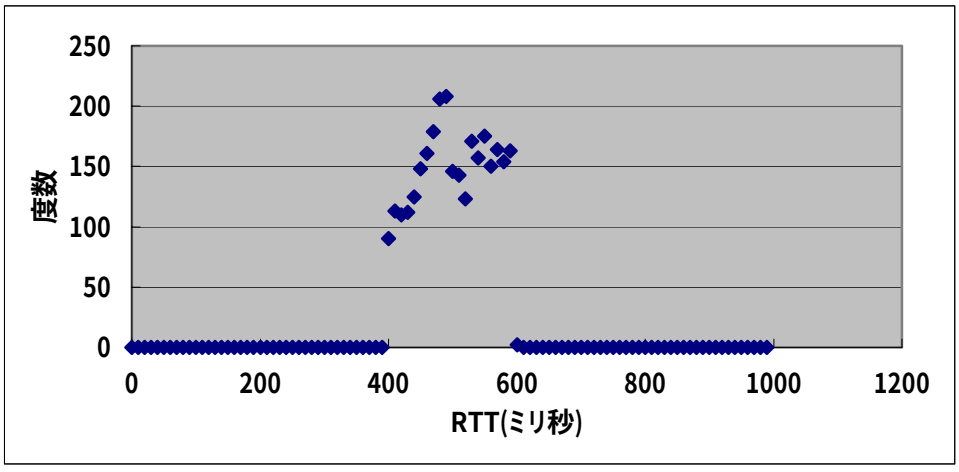

図 2: 遅延として一様分布を用いた場合の遅延の頻度分布
(設定: 遅延 400-600msec, 送信間隔 0.1 秒)

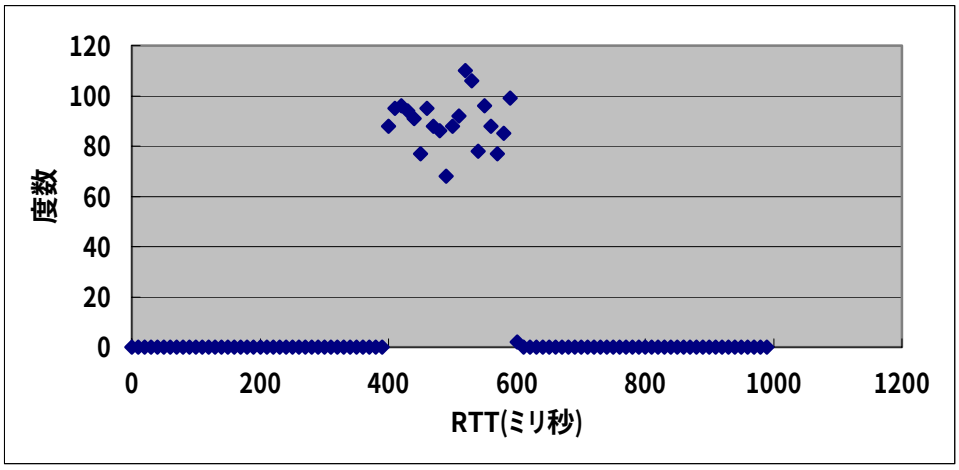

図 3: 遅延として一様分布を用いた場合の遅延の頻度分布
(設定: 遅延 400-600msec, 送信間隔 1 秒)

□ 遅延として正規分布を用いた場合の効果

遅延として正規分布を用いた場合、遅延は平均値を中心に山形の分布となります。例えば、本キットが提供する API を使って下のように設定した場合、遅延は 500 ミリ秒を中心とした山形の揺らぎとなって現れます。

```
EDN_pipe(10, EDN_PIPE_CONFIG, 0.0,
        EDN_PIPE_ETYPE_ERR, 0.0, 0.0,
        EDN_PIPE_DTYPE_NDELAY, 500.0, 100.0);
```

実際にブリッジ設定の片道に対し上記のような設定を行ない、0.1 秒ごとに 3000 回 ICMP Echo を送信したときの RTT の頻度分布を図 4 に示します。この図から遅延の度数分布は 500 ミリ秒を中心に山形となっていることが読み取れます。

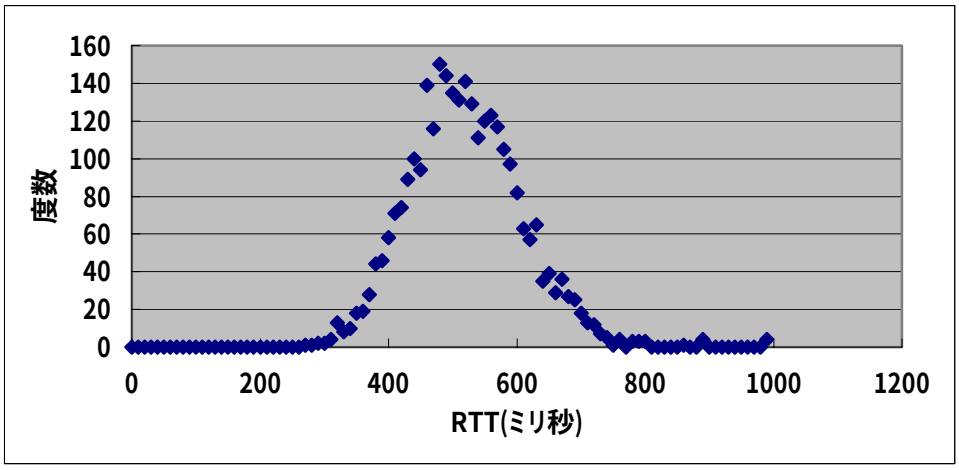

図 4: 遅延として正規分布を用いた場合の遅延の頻度分布
(設定: 平均遅延 500msec, 分散 100msec)

□ 遅延として対数正規分布を用いた場合の効果

遅延として対数正規分布を用いた場合、遅延は exp(m+σ*σ/2)を平均とした右がなだらかな山形の分布となりって現れます。例えば、本キットが提供する API を使って下のように設定した場合、遅延は約 500 ミリ秒を平均として右が緩やかな山形の揺らぎとなって現れます。

```
EDN_pipe(10, EDN_PIPE_CONFIG, 0.0,
        EDN_PIPE_ETYPE_ERR, 0.0, 0.0,
        EDN_PIPE_DTYPE_LDELAY, 5.705448, 1.0);
```

実際にブリッジ設定の片道に対し上記のような設定を行ない、0.1 秒ごとに 3000 回 ICMP Echo を送信したときの RTT の頻度分布を図 5 に示します。この図から遅延は 500 ミリ秒より少し大きな値を平均として変化しているとことが読み取れます。また、分布が対数正規分布ではなく、遅延が大きくなるほど度数も大きくなっています。これは 0.1 秒ごとに ICMP Echo を送信しているため、遅延の揺らぎのために本来小さな遅延となったはずのパケットが前のパケットを追い越すことが出来ず、結果大きな遅延として送信されることに起因しています。この問題は送信間隔を十分にとることによって解決できます。送信間隔を 1 秒に設定した時の頻度分布を図 6 に示します。こちらの図では分布はほぼ対数正規分布に従っています。

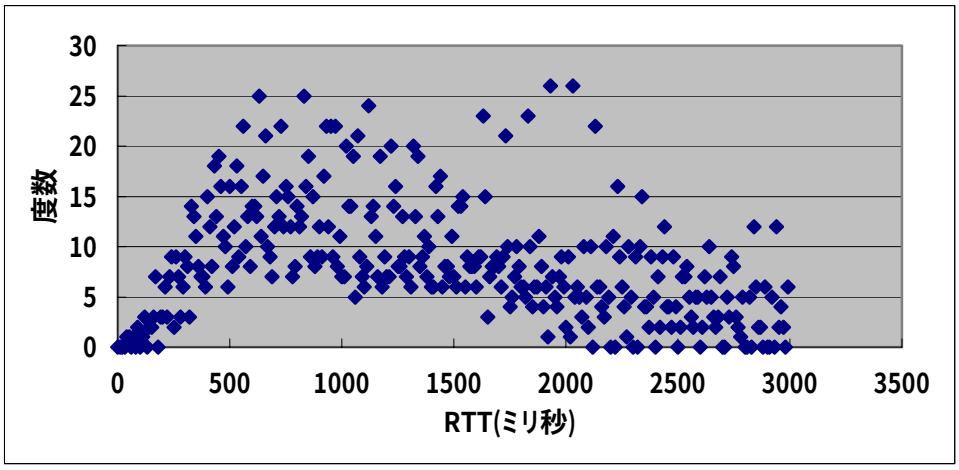

図 5: 遅延として一様分布を用いた場合の遅延の頻度分布
(設定: 遅延平均 500msec, 送信間隔 0.1 秒)

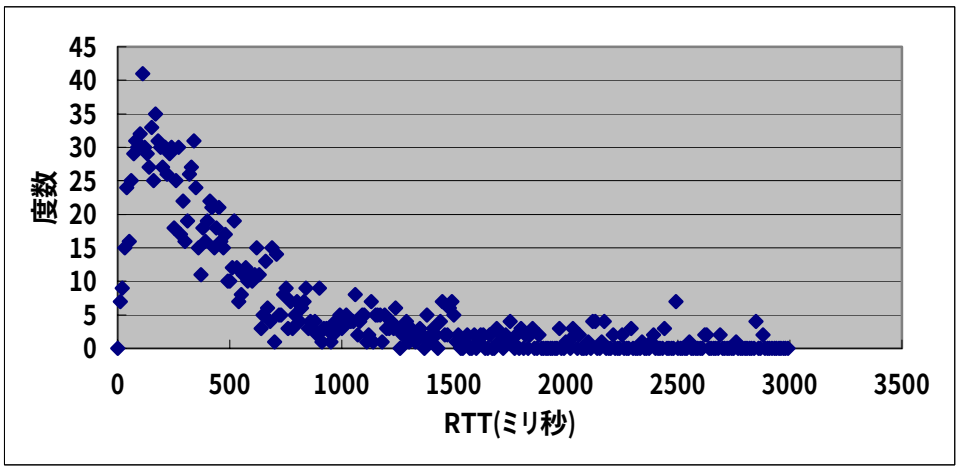

図 6: 遅延として一様分布を用いた場合の遅延の頻度分布
(設定: 遅延平均 500msec, 送信間隔 1 秒)

□ 伝送エラー率として一定値を用いた場合の効果

伝送エラー率として一定値を用いた場合、設定した伝送エラー率が実際の伝送エラー率として現れます。例えば、本キットが提供する API を使って下のように設定した場合、伝送エラー率は 10%となります。

```
EDN_pipe(10, EDN_PIPE_CONFIG, 0.0,
        EDN_PIPE_ETYPE_ERR, 0.1, 0.0,
        EDN_PIPE_DTYPE_DELAY, 0.0, 0.0);
```

弊社において実際にブリッジ設定の片道に対し上記のような設定を行ない、0.1 秒ごとに 1000 回 ICMP Echo を送信したときの伝送エラー率は 10%でした。

□ 伝送エラー率として一様分布を用いた場合の効果

伝送エラー率として一様分布を用いた場合、設定した伝送エラー率の上限値と下限値の中間値が実際の伝送エラー率として現れます。例えば、本キットが提供する API を使って下のように設定した場合、伝送エラー率は 10%となります。

```
EDN_pipe(10, EDN_PIPE_CONFIG, 0.0,
        EDN_PIPE_ETYPE_UERR, 0.05, 0.15,
        EDN_PIPE_DTYPE_DELAY, 0.0, 0.0);
```

弊社において実際にブリッジ設定の片道に対し上記のような設定を行ない、0.1 秒ごとに 1000 回 ICMP Echo を送信したときの伝送エラー率は 9%でした。

□ 伝送エラー率として正規分布を用いた場合の効果

伝送エラー率として正規分布を用いた場合、設定した平均伝送エラー率が実際の伝送エラー率として現れます。例えば、本キットが提供する API を使って下のように設定した場合、伝送エラー率は 10%となります。

```
EDN_pipe(10, EDN_PIPE_CONFIG, 0.0,
        EDN_PIPE_ETYPE_NERR, 0.10, 0.02,
        EDN_PIPE_DTYPE_DELAY, 0.0, 0.0);
```

弊社において実際にブリッジ設定の片道に対し上記のような設定を行ない、0.1 秒ごとに 1000 回 ICMP Echo を送信したときの伝送エラー率は 10%でした。

□ 伝送エラー率として対数正規分布を用いた場合の効果

伝送エラー率として対数正規分布を用いた場合、指定した m(e1)、σ(e2)より計算される exp(m+σ*σ/2)が実際の伝送エラー率として現れます。例えば、本キットが提供する API を使って下のように設定した場合、伝送エラー率は 10%となります。

```
EDN_pipe(10, EDN_PIPE_CONFIG, 0.0,
        EDN_PIPE_ETYPE_UERR, 0.05, 0.15,
        EDN_PIPE_DTYPE_DELAY, 0.0, 0.0);
```

弊社において実際にブリッジ設定の片道に対し上記のような設定を行ない、0.1 秒ごとに 1000 回 ICMP Echo を送信したときの伝送エラー率は 10%でした。

お問合せ先:
株式会社インターネットオートモビリティ研究所
東京都港区西新橋1-4-12 ルート西新橋ビル7F
TEL: 03-5511-6691　FAX: 03-5511-6692　E-Mail: info@ial.jp