

ipgen - Interactive Packet Generator

<https://github.com/iij/ipgen/>

This presentation material was written by

ryo@iij.ad.jp

Presented by msaitoh@n.o

What is ipgen?

- ipgen is a packet generator, benchmark, performance measurement tool.

Motivation

- We are developing router.
- We must do performance test over and over to tuning IP forwarding and to do tests.
- in so doing, we use various benchmark programs/boxes.

- ftp, ping -f, iperf, ttcp, nuttcp, etc...
 - simple and convenient :-)
 - sometime they cannot achieve enough performance depending on hardware and/or network stack implementation :-(
 - cannot get good accuracy and detailed information of results :-(
- Proprietary router tester products (SPIRENT communications, Ixia, Artiza Networks, etc...)
 - High reliable benchmarks :-)
 - very expensive :-(
 - device busy. most of the time, someone use it. (EBUSY) :-(

- Want to test easily on my desk!
 - It's required to speed-up NetBSD MP network stack project.
- with reasonable performance, good accuracy and detailed information of results

ryo@ made it.

```
ipgen - interactive packet generator ver1.21 - netmap API:11
Interface: igb3      < <<< Interface: igb1

Total Count:
TX:                  0 pkt      TX:                  58171736 pkt
TX-etc:              0 pkt      TX-etc:              0 pkt
TX-underrun:         0 pkt      TX-underrun:         58382 pkt
RX:                  58172166 pkt  RX:                  0 pkt
RX-drop:             0 pkt      RX-drop:             0 pkt
RX-dup:              0 pkt      RX-dup:              0 pkt
RX-reorder:          0 pkt      RX-reorder:          0 pkt
RX-reorder/flow:    0 pkt      RX-reorder/flow:    0 pkt
RX-flowctrl:        0 pkt      RX-flowctrl:        0 pkt
RX-arp:              0 pkt      RX-arp:              0 pkt
RX-icmpocho:         0 pkt      RX-icmpocho:         0 pkt
  icmpunreach:       0 pkt      icmpunreach:         0 pkt
  icmpredirect:      0 pkt      icmpredirect:        0 pkt
  icmpother:         0 pkt      icmpother:           0 pkt
RX-other:            0 pkt      RX-other:            0 pkt

Delta:
TX:                  0 pps      TX:                  1486854 pps
TX:                  0 bytes/s   TX:                  124895736 bytes/s
TX: 0.000000000 Mbps   TX: 999.1658888 Mbps
RX:                  1486723 pps  RX:                  0 pps
RX:                  124884732 bytes/s  RX:                  0 bytes/s
RX: 999.8778568 Mbps   RX: 0.000000000 Mbps

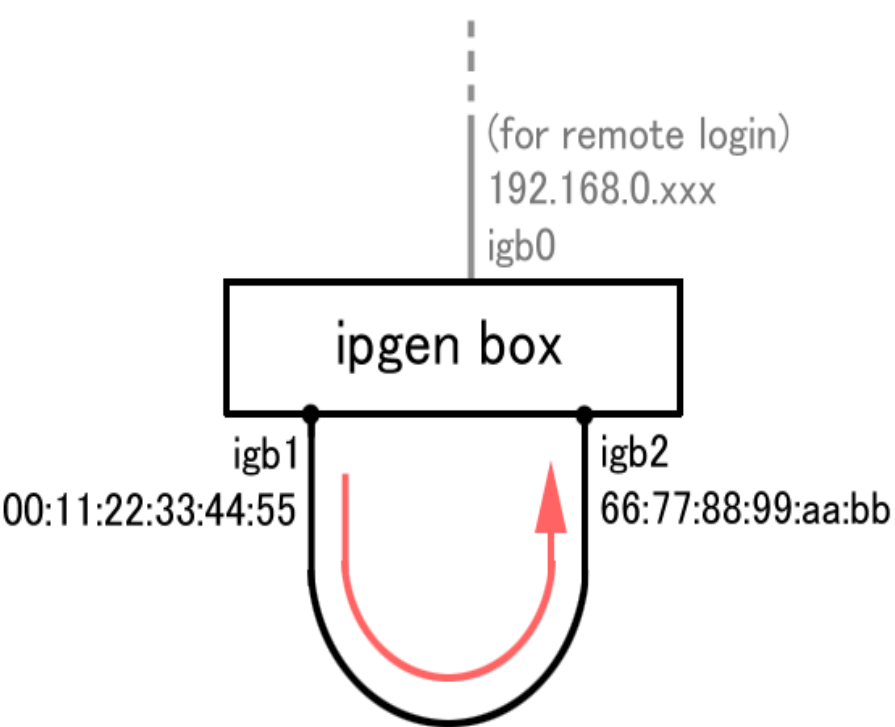
Latency:  min: 0.869373888 ns    min: 0.000000000 ns
           max: 1.285783888 ns    max: 0.000000000 ns
           avg: 0.735213222 ns    avg: 0.000000000 ns

Control:
  Hz: 1000      Flow:[10      ]      Traffic: Burst[*]/Steady[ ]
TX-control:
  TX-pktsize:[46      ]      TX-pktsize:[46      ]
  TX-pps: [10      ]      TX-pps: [1488895 ]
  (max sustained:18      )      (max sustained:1488895 )
  Mbps: 0.000000000      Mbps: 999.999848
  Start[ ]/Stop[*]      Start[*]/Stop[ ]

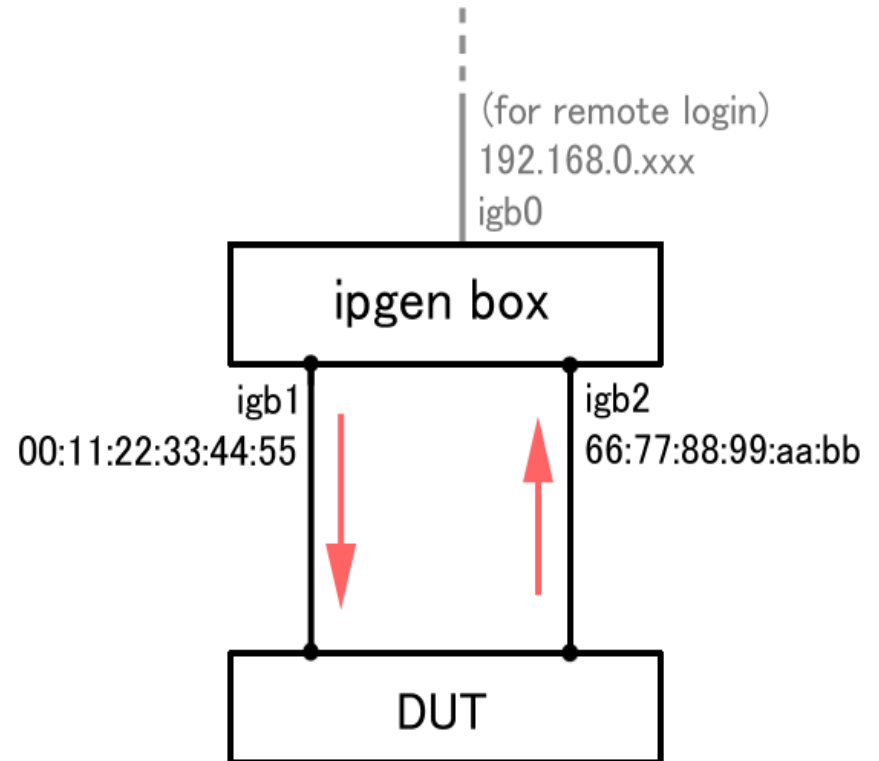
USAGE:
'z' - clear statistics, 'q' - quit
<TAB>,<ARROW>,<N>,<P>,<F>,<B> - select, <ENTER> - edit, <ESC> - cancel
```

example

loopback test



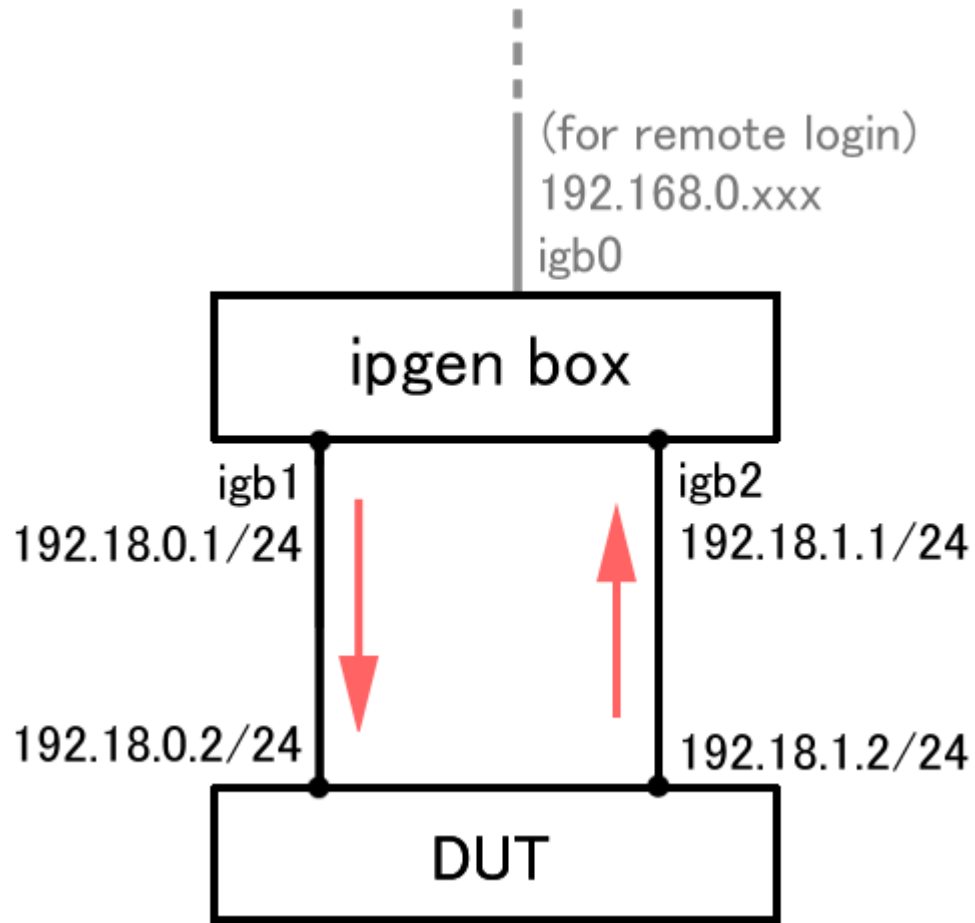
bridge test (L2 forwarding)



```
# ipgen -T igb1,66:77:88:99:aa:bb -R igb2,00:11:22:33:44:55
```

example

L3 forwarding test



```
# ipgen -T igb1,198.18.0.2,198.18.0.1/24 -R igb2,192.18.1.2,192.18.1.1/24
```

using FreeBSD netmap

Luigi-san has already written simple 'pkt-gen' program in FreeBSD:tools/tools/netmap







[/\[base\]](#) / [/head](#) / [/tools](#) / [/tools](#) / netmap

Index of /head/tools/tools/netmap

Files shown: 5

Directory revision: [281746](#) (of [296448](#))

Sticky Revision:

File ^	Rev.	Age	Author	Last log entry
 Parent Directory				
 Makefile	264400	22 months	imp	NO_MAN= has been deprecated in favor of MAN= for
 README	261909	2 years	luigi	This new version of netmap brings you the following: -
 bridge.c	261909	2 years	luigi	This new version of netmap brings you the following: -
 pkt-gen.c	281746	10 months	adrian	Update pkt-gen to optionally use randomised source/c
 vare-ctl.c	270063	18 months	luigi	Update to the current version of netmap. Mostly bugfix

!!  I wrote using examples from it :)

Cutting corners to meet arbitrary management deadlines



Essential

Copying and Pasting from Stack Overflow

O'REILLY®

The Practical Developer
@ThePracticalDev

copy and paste is the best way to
programming.

Features

- Interactive UI
- Drop/Duplicate/Reorder counter
- Multiple flows support
- Inter Packet Gap support
- RFC2544 test
- IPv6 support

ipgen check reordering with considering each flow

[A→B #1]				[A→B #1]
[A→C #2]				[A→B #3]
[A→B #3]	→	[DUT]	→	[A→B #5]
[A→C #4]				[A→C #2]
[A→B #5]				[A→C #4]
[A→C #6]				[A→C #6]

#1→#3→#5→#2→#4→#6 ... is this reordered?

this is reordered totally, but

this is not reordered per flow.

ipgen check reordering with considering each flow

[A→B #1 ##1]					[A→B #1 ##1]
[A→C #2 ##1]					[A→B #3 ##2]
[A→B #3 ##2]	→	[DUT]	→		[A→B #5 ##3]
[A→C #4 ##2]					[A→C #2 ##1]
[A→B #5 ##3]					[A→C #4 ##2]
[A→C #6 ##3]					[A→C #6 ##3]

##1→##2→##3→##1→##2→##3

for each flow, not reordered.

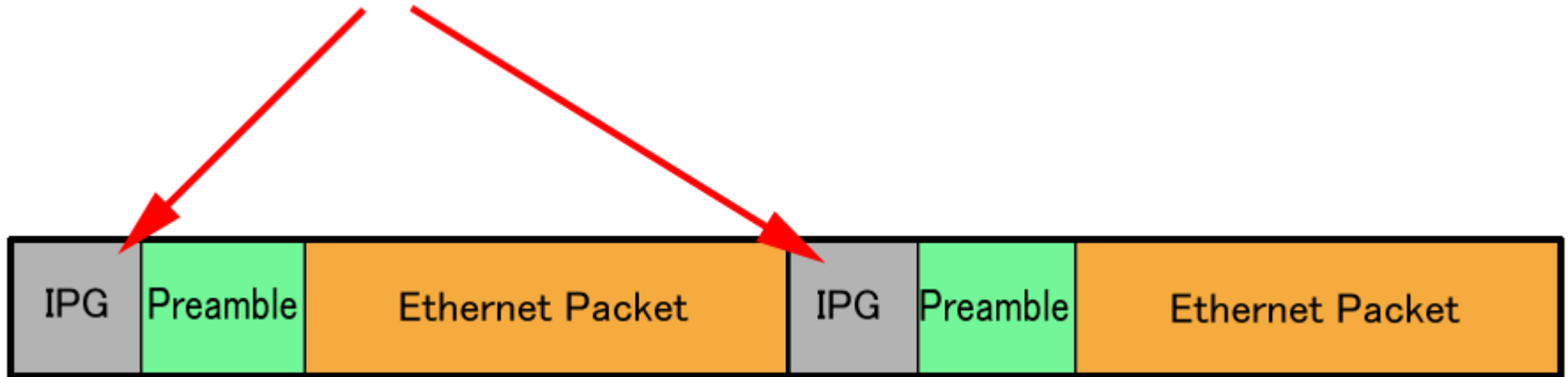
Burst transmission problem

- E.g., sending packets with 1,000 pps
 - if the granularity of the internal timer is 1 sec., 1,000 packets are sent in bulk at the beginning in each time slot and remaining period of the slot will be idle
- Finer timer solves the problem?
 - No. 1,000 Hz timer can solve the problem on 1,000 pps, but cannot solve on 100,000 pps
- More finer timer?



Inter Packet Gap (aka Inter Frame Gap)

- What is IPG?



*Ethernet devices must allow a minimum idle period between transmission of Ethernet packets.
(from wikipedia)*

Inter Packet Gap is the idle period.

on most ethernet device, IPG is configurable.
also Intel's GbE can!



Integrated GbE Controller PRM—Programming Interface—
C2000 Product Family

6.12.3 Transmit IPG Register—TIPG (0x0410; R/W)

This register controls the Inter Packet Gap (IPG) timer.

Field	Bit(s)	Initial Value	Description
IPGT	9:0	0x08	<p>IPG Back to Back</p> <p>Specifies the IPG length for back to back transmissions in both full and half duplex. Measured in increments of the MAC clock:</p> <p>8 ns MAC clock when operating @ 1 Gb/s. 80 ns MAC clock when operating @ 100 Mb/s. 800 ns MAC clock when operating @ 10 Mb/s.</p> <p>IPGT specifies the IPG length for back-to-back transmissions in both full duplex and half duplex. Note that an offset of 4 byte times is added to the programmed value to determine the total IPG. As a result, a value of 8 is recommended to achieve a 12 byte time IPG.</p>
IPGR1	19:10	0x04	<p>IPG Part 1</p> <p>Specifies the portion of the IPG in which the transmitter defers to receive events. IPGR1 should be set to 2/3 of the total effective IPG (8).</p> <p>Measured in increments of the MAC clock:</p> <p>8 ns MAC clock when operating @ 1 Gb/s. 80 ns MAC clock when operating @ 100 Mb/s 800 ns MAC clock when operating @ 10 Mb/s.</p>
IPGR	29:20	0x06	<p>IPG After Deferral</p> <p>Specifies the total IPG time for non back-to-back transmissions (transmission following deferral) in half duplex.</p> <p>Measured in increments of the MAC clock:</p> <p>8 ns MAC clock when operating @ 1 Gb/s. 80 ns MAC clock when operating @ 100 Mb/s 800 ns MAC clock when operating @ 10 Mb/s.</p> <p>An offset of 5-byte times must be added to the programmed value to determine the total IPG after a defer event. A value of 7 is recommended to achieve a 12-byte effective IPG. Note that the IPGR must never be set to a value greater than IPGT. If IPGR is set to a value equal to or larger than IPGT, it overrides the IPGT IPG setting in half duplex resulting in inter-packet gaps that are larger than intended by IPGT. In this case, full duplex is unaffected and always relies on IPGT.</p>
Reserved	31:30	00b	<p>Reserved</p> <p>Write 0, ignore on read.</p>

but no API to configure IPG from userland.
I wrote small patch!

Index: if_igb.c

```
-----  
--- if_igb.c (revision 292398)  
+++ if_igb.c (working copy)  
@@ -547,6 +548,12 @@  
    }  
}  
  
+   SYSCTL_ADD_PROC(device_get_sysctl_ctx(dev),  
+   SYSCTL_CHILDREN(device_get_sysctl_tree(dev)),  
+   OID_AUTO, "tipg", CTLTYPE_INT|CTLFLAG_RW,  
+   adapter, 0, igb_sysctl_tipg, "I",  
+   "Transmit IPG register");  
+  
+   /*  
+   ** Start from a known state, this is  
+   ** important in reading the nvm and  
@@ -6377,3 +6384,23 @@  
   IGB_CORE_UNLOCK(adapter);  
   return (0);  
}  
+  
+static int  
+igb_sysctl_tipg(SYSCTL_HANDLER_ARGS)  
+{  
+   struct adapter *adapter = (struct adapter *)arg1;  
+   int error, value;  
+   u32 reg;  
+  
+   value = E1000_READ_REG(&adapter->hw, E1000_TIPG) & E1000_TIPG_IPGT_MASK;  
+   error = sysctl_handle_int(oidp, &value, 0, req);  
+   if (error || req->newptr == NULL)  
+       return (error);  
+  
+   reg = E1000_READ_REG(&adapter->hw, E1000_TIPG);  
+   reg &= ~E1000_TIPG_IPGT_MASK;  
+   reg |= value & E1000_TIPG_IPGT_MASK;  
+   E1000_WRITE_REG(&adapter->hw, E1000_TIPG, reg);  
+  
+   return (0);  
+}
```

This patch can control IPG by sysctl(8)

```
# sysctl dev.igb | grep tipg
```

```
dev.igb.5.tipg: 8
```

```
dev.igb.4.tipg: 8
```

```
dev.igb.3.tipg: 8
```

```
dev.igb.2.tipg: 8
```

```
dev.igb.1.tipg: 8
```

```
dev.igb.0.tipg: 8
```

Controlling IPG can provide steady traffic

no IPG adjustment (burst traffic mode)



with IPG adjustment (steady traffic mode)



RFC2544 test

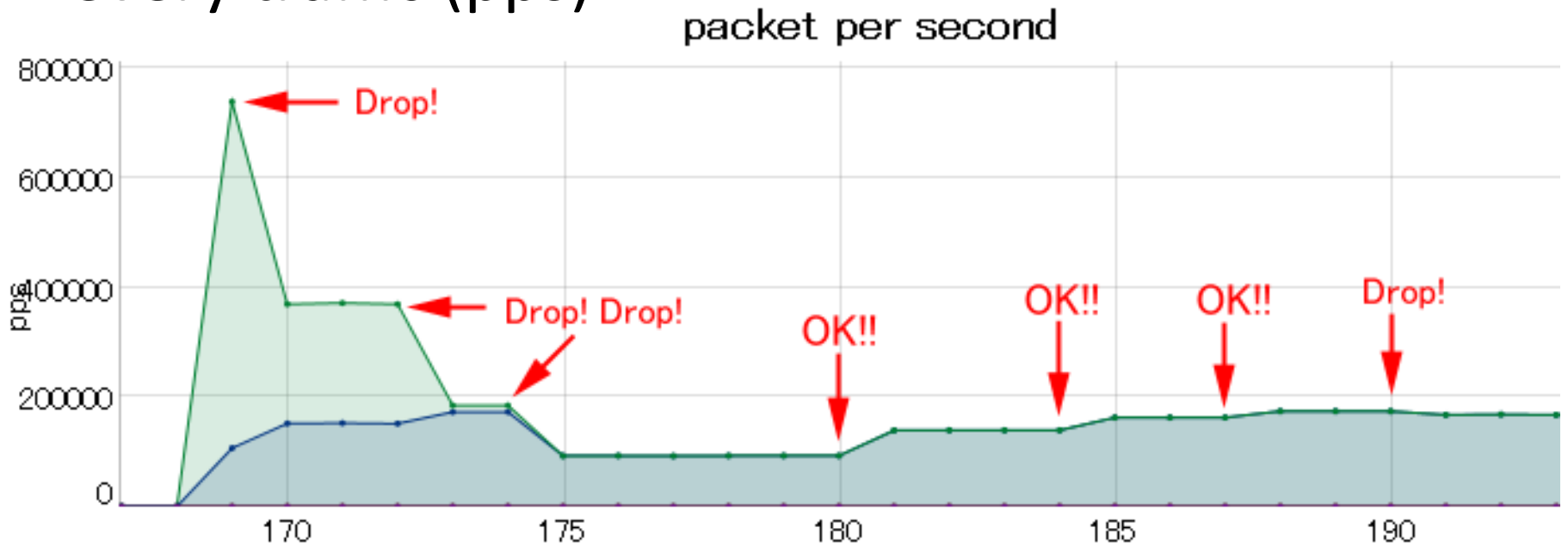
Q. What is RFC2544 test?

A. SEE RFC2544 :)

The objective of the test is to determine the minimum interval between bursts which the DUT can process with no frame loss.

ipgen supports RFC2544 test mode.

It does binary search to avoid measuring with every traffic (pps)



An example of binary search. If packet drops happen, the offered traffic decreases. If no packet is lost, the offered traffic increases.

Result of ipgen RFC2544 test mode

```
# ipgen --rfc2544 -T igb2,00:60:e0:5c:4e:e7 \  
-R igb4,00:60:e0:5c:4e:e5
```

```
# ipgen -T igb2,00:60:e0:5c:4e:e7 -R igb4,00:60:e0:5c:4e:e5 --rfc2544 --rfc2544-trial-duration 0  
ipgen v1.21  
igb(4) TIPG feature supported  
HZ=1000  
igb2 -> igb4, IP pktsize 46, 1488095 pps, 999 Mbps (999999840 bps)  
igb2: waiting link up.....OK  
igb4: waiting link up.OK  
igb4(00:60:e0:5c:4e:e7) -> 00:60:e0:5c:4e:e5  
igb2(00:60:e0:5c:4e:e5) -> 00:60:e0:5c:4e:e7  
Exiting...
```

```
framesize|0M 100M 200M 300M 400M 500M 600M 700M 800M 900M 1Gbps
```

64	#####	105.71Mbps ,	157310/1488095pps
128	#####	93.13Mbps ,	78654/ 844594pps
256	#####	276.96Mbps ,	125433/ 452898pps
512	#####	650.38Mbps ,	152814/ 234962pps
1024	#####	999.99Mbps ,	119731/ 119731pps
1280	#####	999.99Mbps ,	96153/ 96153pps
1408	#####	1000.00Mbps ,	87535/ 87535pps
1518	#####	1000.00Mbps ,	81274/ 81274pps

```
framesize|0 |100k|200k|300k|400k|500k|600k|700k|800k|900k|1.0m|1.1m|1.2m|1.3m|1.4m|1.5m pps
```

64	#####	157310/1488095pps ,	10.57%
128	###	78654/ 844594pps ,	9.31%
256	#####	125433/ 452898pps ,	27.70%
512	#####	152814/ 234962pps ,	65.04%
1024	#####	119731/ 119731pps ,	100.00%
1280	#####	96153/ 96153pps ,	100.00%
1408	#####	87535/ 87535pps ,	100.00%
1518	#####	81274/ 81274pps ,	100.00%

CONCLUSION

- We made easy-to-use packet generator
- RFC2544 test supported
- netmap is very cool!
I hope someone to port netmap to NetBSD :)