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 -(E)
  – 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.
example

loopback test

bridge test (L2 forwarding)

```bash
```
# 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

I wrote using examples from it :)
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
drop/duplicate/reorder counter

each packet has a sequence.

ipgen has a bitmap flag internally.

<table>
<thead>
<tr>
<th>#seq</th>
<th>received bitmap</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>111111111111111111111111111111111111111111111111111111111111111</td>
</tr>
<tr>
<td>50</td>
<td>111111111111111111111111111111111111111111111111111111111111111</td>
</tr>
<tr>
<td>100</td>
<td>111111111111111111111111111111111111111111111111111111111111111</td>
</tr>
<tr>
<td>150</td>
<td>000000000000000000000000000000000000000000000000000000000000000</td>
</tr>
<tr>
<td>200</td>
<td>000000000000000000000000000000000000000000000000000000000000000</td>
</tr>
<tr>
<td>250</td>
<td>000000000000000000000000000000000000000000000000000000000000000</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
#105→#106→#108→#109→#110→

100: 111111101111110000000000000000000000000000000000000
   ~
   #107 dropped?

#105→#106→#108→#109→#110→#107→#111…
   ~~~~

100: 111111111111111000000000000000000000000000000000000
   ~
   #107 not dropped. (reordered)

#105→#106→#108→#109→#110→#107→#111→#108→
   ~~~~
   ~~~~

100: 111111111111111100000000000000000000000000000000000
   ~
   #108 duplicate!
ipgen check reordering with considering each flow

\[ A \rightarrow B \ #1 \]
\[ A \rightarrow C \ #2 \]
\[ A \rightarrow B \ #3 \]
\[ A \rightarrow B \ #3 \rightarrow \text{[DUT]} \rightarrow \text{[A→B #5]} \]
\[ A \rightarrow C \ #4 \]
\[ A \rightarrow C \ #4 \]
\[ A \rightarrow B \ #5 \]
\[ A \rightarrow B \ #5 \]
\[ A \rightarrow C \ #6 \]
\[ A \rightarrow C \ #6 \]

#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**

<table>
<thead>
<tr>
<th>Flow</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A→B</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A→C</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A→B</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A→B</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A→C</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A→C</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

##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!

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

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit(s)</th>
<th>Initial Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPGT</td>
<td>9:0</td>
<td>0x08</td>
<td>IPG Back to Back. Specifies the IPG length for back to back transmissions in both full and half duplex. Measured in increments of the MAC clock: 8 ns MAC clock when operating @ 1 Gbps. 80 ns MAC clock when operating @ 100 Mbps. 800 ns MAC clock when operating @ 10 Mbps. IPG 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.</td>
</tr>
<tr>
<td>IPGR1</td>
<td>19:10</td>
<td>0x04</td>
<td>IPG Part 1. Specifies the portion of the IPG in which the transmitter defers to receive events. IPGR should be set to 2/3 of the total effective IPG (8). Measured in increments of the MAC clock: 8 ns MAC clock when operating @ 1 Gbps. 80 ns MAC clock when operating @ 100 Mbps. 800 ns MAC clock when operating @ 10 Mbps.</td>
</tr>
<tr>
<td>IPGR</td>
<td>29:20</td>
<td>0x06</td>
<td>IPG After Deferral. Specifies the total IPG time for non back-to-back transmissions (transmission following deferral) in half duplex. Measured in increments of the MAC clock: 8 ns MAC clock when operating @ 1 Gbps. 80 ns MAC clock when operating @ 100 Mbps. 800 ns MAC clock when operating @ 10 Mbps. 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 that IPGT, it overrides the IPGT IPG setting in full duplex resulting in inter-packet gaps that are larger then intended by IPGT. In this case, full duplex is unaffected and always relies on IPGT.</td>
</tr>
<tr>
<td>Reserved</td>
<td>31:30</td>
<td>00b</td>
<td>Reserved. Write 0, ignore on read.</td>
</tr>
</tbody>
</table>
but no API to configure IPG from userland.
I wrote small patch!

```c
Index: if_igb.c
============================================================================
--- if_igb.c  (revision 292398)
+++ if_igb.c  (working copy)
@@ -547,6 +548,12 @@
 }

+SYSCALL_ADD_PROC(device_get_sysctl_ctx(dev),
+       SYSCALL_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(nidp, &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 \texttt{ipgen RFC2544} test mode

\texttt{# ipgen --rfc2544} -T \texttt{igb2,00:60:e0:5c:4e:e7} \(\backslash\)
\texttt{-R igb4,00:60:e0:5c:4e:e5}

\texttt{# ipgen} v1.21
\texttt{igb(4) TIPG feature supported}
\texttt{HZ=1000}
\texttt{igb2 -> igb4, IP pktsize 46, 1488095 pps, 999 Mbps (999999840 bps)}
\texttt{igb2: waiting link up.......OK}
\texttt{igb4: waiting link up.OK}
\texttt{igb4(00:60:e0:5c:4e:e7) -> 00:60:e0:5c:4e:e5}
\texttt{igb2(00:60:e0:5c:4e:e5) -> 00:60:e0:5c:4e:e7}
\texttt{Exiting...}

framesize|0M|100M|200M|300M|400M|500M|600M|700M|800M|900M|1Gbps
---|---|---|---|---|---|---|---|---|---|--
64 | SIDEBAR| 105.71Mbps, 157310/1488095pps
120 | SIDEBAR| 93.13Mbps, 78654/ 844594pps
256 | SIDEBAR| 276.96Mbps, 125433/ 452890pps
512 | SIDEBAR| 650.38Mbps, 152814/234962pps
1024 | SIDEBAR| 999.99Mbps, 119731/119731pps
1200 | SIDEBAR| 999.99Mbps, 96153/ 96153pps
1400 | SIDEBAR| 1000.00Mbps, 87535/ 87535pps
1518 | SIDEBAR| 1000.00Mbps, 81274/ 81274pps

framesize|0k|100k|200k|300k|400k|500k|600k|700k|800k|900k|1.0m|1.1m|1.2m|1.3m|1.4m|1.5m
---|---|---|---|---|---|---|---|---|---|---|--|--|--|--|--|--|--|
64 | SIDEBAR| 157310/1488095pps, 10.57%
120 | SIDEBAR| 78654/ 844594pps, 9.31%
256 | SIDEBAR| 125433/ 452890pps, 27.70%
512 | SIDEBAR| 152814/234962pps, 65.04%
1024 | SIDEBAR| 119731/119731pps, 100.00%
1200 | SIDEBAR| 96153/ 96153pps, 100.00%
1400 | SIDEBAR| 87535/ 87535pps, 100.00%
1518 | SIDEBAR| 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 :)