HOWTO for the linux packet generator

Enable CONFIG_NET_PKTGEN to compile and build pktgen either in-kernel or as a module. A module is preferred; modprobe pktgen if needed. Once running, pktgen creates a thread for each CPU with affinity to that CPU. Monitoring and controlling is done via /proc. It is easiest to select a suitable sample script and configure that.

On a dual CPU:

```
ps aux | grep pkt
root       129  0.3  0.0     0    0 ?        SW    2003 523:20 [kpktgend_0]
root       130  0.3  0.0     0    0 ?        SW    2003 509:50 [kpktgend_1]
```

For monitoring and control pktgen creates:

```
/proc/net/pktgen/pgctrl
/proc/net/pktgen/kpktgend_X
/proc/net/pktgen/ethX
```

Tuning NIC for max performance

The default NIC settings are (likely) not tuned for pktgen's artificial overload type of benchmarking, as this could hurt the normal use-case.

Specifically increasing the TX ring buffer in the NIC:

```
# ethtool -G ethX tx 1024
```

A larger TX ring can improve pktgen's performance, while it can hurt in the general case, 1) because the TX ring buffer might get larger than the CPU's L1/L2 cache, 2) because it allows more queueing in the NIC HW layer (which is bad for bufferbloat).

One should hesitate to conclude that packets/descriptors in the HW TX ring cause delay. Drivers usually delay cleaning up the ring-buffers for various performance reasons, and packets stalling the TX ring might just be waiting for cleanup.

This cleanup issue is specifically the case for the driver ixgbe (Intel 82599 chip). This driver (ixgbe) combines TX+RX ring cleanups, and the cleanup interval is affected by the ethtool --coalesce setting of parameter "rx-usecs".

For ixgbe use e.g. "30" resulting in approx 33K interrupts/sec (1/30*10^6):

```
# ethtool -C ethX rx-usecs 30
```

Kernel threads

Pktgen creates a thread for each CPU with affinity to that CPU. Which is controlled through procfile /proc/net/pktgen/kpktgend_X.

Example: /proc/net/pktgen/kpktgend_0:

```
Running:
Stopped: eth4@0
Result: OK: add_device=eth4@0
```

Most important are the devices assigned to the thread.

The two basic thread commands are:

- add_device DEVICE@NAME -- adds a single device
- rem_device_all -- remove all associated devices

When adding a device to a thread, a corresponding procfile is created which is used for configuring this device. Thus, device names need to be unique.

To support adding the same device to multiple threads, which is useful with multi queue NICs, the device naming scheme is extended with "@": device@something

The part after "@" can be anything, but it is custom to use the thread number.

Viewing devices

The Params section holds configured information. The Current section holds running statistics. The Result is printed after a run or after interruption. Example:

```
/proc/net/pktgen/eth4@0

Params: count 100000  min_pkt_size: 60  max_pkt_size: 60
    frags: 0  delay: 0  clone_skb: 64  ifname: eth4@0
    flows: 0 flowlen: 0
    queue_map_min: 0  queue_map_max: 0
    dst_min: 192.168.81.2  dst_max:
```

```
    src_min:   src_max:
    src_mac: 90:e2:ba:0a:56:b4 dst_mac: 00:1b:21:3c:9d:f8
    udp_src_min: 9  udp_src_max: 109  udp_dst_min: 9  udp_dst_max: 9
    src_mac_count: 0  dst_mac_count: 0
    Flags: UDPSRC_RND  NO_TIMESTAMP  QUEUE_MAP_CPU
Current:
    pkts-sofar: 100000  errors: 0
    started: 623913381008us  stopped: 623913396439us idle: 25us
    seq_num: 100001  cur_dst_mac_offset: 0  cur_src_mac_offset: 0
    cur_saddr: 192.168.8.3  cur_daddr: 192.168.81.2
    cur_udp_dst: 9  cur_udp_src: 42
    cur_queue_map: 0
    flows: 0
Result: OK: 15430(c15405+d25) usec, 100000 (60byte,0frags)
6480562pps 3110Mb/sec (3110669760bps) errors: 0
```

Configuring devices

This is done via the /proc interface, and most easily done via pgset as defined in the sample scripts. You need to specify PGDEV environment variable to use functions from sample scripts, i.e.:

```
export PGDEV=/proc/net/pktgen/eth4@0
source samples/pktgen/functions.sh
```

Examples:

```
pg_ctrl start           starts injection.
pg_ctrl stop            aborts injection. Also, ^C aborts generator.

pgset "clone_skb 1"     sets the number of copies of the same packet
pgset "clone_skb 0"     use single SKB for all transmits
pgset "burst 8"         uses xmit_more API to queue 8 copies of the same
                        packet and update HW tx queue tail pointer once.
                        "burst 1" is the default
pgset "pkt_size 9014"   sets packet size to 9014
pgset "frags 5"         packet will consist of 5 fragments
pgset "count 200000"    sets number of packets to send, set to zero
                        for continuous sends until explicitly stopped.

pgset "delay 5000"      adds delay to hard_start_xmit(). nanoseconds

pgset "dst 10.0.0.1"    sets IP destination address
                        (BEWARE! This generator is very aggressive!)

pgset "dst_min 10.0.0.1"            Same as dst
pgset "dst_max 10.0.0.254"          Set the maximum destination IP.
pgset "src_min 10.0.0.1"            Set the minimum (or only) source IP.
pgset "src_max 10.0.0.254"          Set the maximum source IP.
pgset "dst6 fec0::1"     IPV6 destination address
pgset "src6 fec0::2"     IPV6 source address
pgset "dstmac 00:00:00:00:00:00"    sets MAC destination address
pgset "srcmac 00:00:00:00:00:00"    sets MAC source address

pgset "queue_map_min 0" Sets the min value of tx queue interval
pgset "queue_map_max 7" Sets the max value of tx queue interval, for multiqueue devices
                        To select queue 1 of a given device,
                        use queue_map_min=1 and queue_map_max=1

pgset "src_mac_count 1" Sets the number of MACs we'll range through.
                        The 'minimum' MAC is what you set with srcmac.

pgset "dst_mac_count 1" Sets the number of MACs we'll range through.
                        The 'minimum' MAC is what you set with dstmac.

pgset "flag [name]"     Set a flag to determine behaviour.  Current flags
                        are: IPSRC_RND # IP source is random (between min/max)
                             IPDST_RND # IP destination is random
                             UDPSRC_RND, UDPDST_RND,
                             MACSRC_RND, MACDST_RND
                             TXSIZE_RND, IPV6,
                             MPLS_RND, VID_RND, SVID_RND
                             FLOW_SEQ,
                             QUEUE_MAP_RND # queue map random
                             QUEUE_MAP_CPU # queue map mirrors smp_processor_id()
                             UDPCSUM,
                             IPSEC # IPsec encapsulation (needs CONFIG_XFRM)
                             NODE_ALLOC # node specific memory allocation
                             NO_TIMESTAMP # disable timestamping
pgset 'flag ![name]'    Clear a flag to determine behaviour.
                        Note that you might need to use single quote in
                        interactive mode, so that your shell wouldn't expand
                        the specified flag as a history command.
```

```
pgset "spi [SPI_VALUE]" Set specific SA used to transform packet.

pgset "udp_src_min 9"   set UDP source port min, If < udp_src_max, then
                        cycle through the port range.

pgset "udp_src_max 9"   set UDP source port max.
pgset "udp_dst_min 9"   set UDP destination port min, If < udp_dst_max, then
                        cycle through the port range.
pgset "udp_dst_max 9"   set UDP destination port max.

pgset "mpls 0001000a,0002000a,0000000a" set MPLS labels (in this example
                                        outer label=16,middle label=32,
                                        inner label=0 (IPv4 NULL)) Note that
                                        there must be no spaces between the
                                        arguments. Leading zeros are required.
                                        Do not set the bottom of stack bit,
                                        that's done automatically. If you do
                                        set the bottom of stack bit, that
                                        indicates that you want to randomly
                                        generate that address and the flag
                                        MPLS_RND will be turned on. You
                                        can have any mix of random and fixed
                                        labels in the label stack.

pgset "mpls 0"           turn off mpls (or any invalid argument works too!)

pgset "vlan_id 77"       set VLAN ID 0-4095
pgset "vlan_p 3"         set priority bit 0-7 (default 0)
pgset "vlan_cfi 0"       set canonical format identifier 0-1 (default 0)

pgset "svlan_id 22"      set SVLAN ID 0-4095
pgset "svlan_p 3"        set priority bit 0-7 (default 0)
pgset "svlan_cfi 0"      set canonical format identifier 0-1 (default 0)

pgset "vlan_id 9999"     > 4095 remove vlan and svlan tags
pgset "svlan 9999"       > 4095 remove svlan tag


pgset "tos XX"           set former IPv4 TOS field (e.g. "tos 28" for AF11 no ECN, default 00)
pgset "traffic_class XX" set former IPv6 TRAFFIC CLASS (e.g. "traffic_class B8" for EF no ECN, default 0(

pgset "rate 300M"        set rate to 300 Mb/s
pgset "ratep 1000000"    set rate to 1Mpps

pgset "xmit_mode netif_receive"  RX inject into stack netif_receive_skb()
                                 Works with "burst" but not with "clone_skb".
                                 Default xmit_mode is "start_xmit".
```

Sample scripts

A collection of tutorial scripts and helpers for pktgen is in the samples/pktgen directory. The helper parameters.sh file support easy and consistent parameter parsing across the sample scripts.

Usage example and help:

```
./pktgen_sample01_simple.sh -i eth4 -m 00:1B:21:3C:9D:F8 -d 192.168.8.2
```

Usage::

```
./pktgen_sample01_simple.sh [-vx] -i ethX

-i : ($DEV)       output interface/device (required)
-s : ($PKT_SIZE)  packet size
-d : ($DEST_IP)   destination IP. CIDR (e.g. 198.18.0.0/15) is also allowed
-m : ($DST_MAC)   destination MAC-addr
-p : ($DST_PORT)  destination PORT range (e.g. 433-444) is also allowed
-t : ($THREADS)   threads to start
-f : ($F_THREAD)  index of first thread (zero indexed CPU number)
-c : ($SKB_CLONE) SKB clones send before alloc new SKB
-n : ($COUNT)     num messages to send per thread, 0 means indefinitely
-b : ($BURST)     HW level bursting of SKBs
-v : ($VERBOSE)   verbose
-x : ($DEBUG)     debug
-6 : ($IP6)       IPv6
-w : ($DELAY)     Tx Delay value (ns)
-a : ($APPEND)    Script will not reset generator's state, but will append its config
```

The global variables being set are also listed. E.g. the required interface/device parameter "-i" sets variable $DEV. Copy the pktgen_sampleXX scripts and modify them to fit your own needs.

Interrupt affinity

Note that when adding devices to a specific CPU it is a good idea to also assign /proc/irq/XX/smp_affinity so that the TX interrupts are bound to the same CPU. This reduces cache bouncing when freeing skbs.

Plus using the device flag QUEUE_MAP_CPU, which maps the SKBs TX queue to the running threads CPU (directly from smp_processor_id()).

Enable IPsec

Default IPsec transformation with ESP encapsulation plus transport mode can be enabled by simply setting:

```
pgset "flag IPSEC"
pgset "flows 1"
```

To avoid breaking existing testbed scripts for using AH type and tunnel mode, you can use "pgset spi SPI_VALUE" to specify which transformation mode to employ.

Current commands and configuration options

Pgcontrol commands:

```
start
stop
reset
```

Thread commands:

```
add_device
rem_device_all
```

Device commands:

```
count
clone_skb
burst
debug

frags
delay

src_mac_count
dst_mac_count

pkt_size
min_pkt_size
max_pkt_size

queue_map_min
queue_map_max
skb_priority

tos           (ipv4)
traffic_class (ipv6)

mpls

udp_src_min
udp_src_max

udp_dst_min
udp_dst_max

node

flag
IPSRC_RND
IPDST_RND
UDPSRC_RND
UDPDST_RND
MACSRC_RND
MACDST_RND
TXSIZE_RND
IPV6
MPLS_RND
VID_RND
SVID_RND
FLOW_SEQ
QUEUE_MAP_RND
QUEUE_MAP_CPU
UDPCSUM
IPSEC
NODE_ALLOC
```

```
NO_TIMESTAMP

spi (ipsec)

dst_min
dst_max

src_min
src_max

dst_mac
src_mac

clear_counters

src6
dst6
dst6_max
dst6_min

flows
flowlen

rate
ratep

xmit_mode <start_xmit|netif_receive>

vlan_cfi
vlan_id
vlan_p

svlan_cfi
svlan_id
svlan_p
```

References:

- ftp://robur.slu.se/pub/Linux/net-development/pktgen-testing/
- ftp://robur.slu.se/pub/Linux/net-development/pktgen-testing/examples/

Paper from Linux-Kongress in Erlangen 2004. - ftp://robur.slu.se/pub/Linux/net-development/pktgen-testing/pktgen_paper.pdf

Thanks to:

Grant Grundler for testing on IA-64 and parisc, Harald Welte, Lennert Buytenhek Stephen Hemminger, Andi Kleen, Dave Miller and many others.

Good luck with the linux net-development.