International Journal of Engineering Inventions
e-ISSN: 2278-7461, p-ISSN: 2319-6491
Volume 3, Issue 5 (December 2013) PP: 16-20

Enhanced Software Based Traffic Generator

Pushpalatha M.
ISE, MVJCE, Bangalore

ABSTRACT: *This Paper provides an overview of the limitations imposed by software based traffic generator used to generate packet on the type of hardware as well as on the latency the operating system create for transmitting a packet. The paper also explains various stages through which a packet flows from the user-space until it is transmitted out from the hardware. Also discusses the architecture, design, implementation and performance between the socket type PF_PACKET, which is used by kernel based traffic generator PKTGEN to generate packets/traffic, also by most of software based traffic generators and a newly created socket type PF_ZERO, whose purpose is for traffic generation better than PF_PACKET by providing a way to generate higher traffic rates when compared to the existing mechanisms.*

Keywords: *pf_packet, pf_zero, pktgen, Software based traffic-generators.*

I. INTRODUCTION

Internet explosion created highly interconnected mesh network connecting every corner of earth, expanding day by day to meet the demands and requirements of many users. As internet expands exponentially the rate at which data is being shared among users has increased tremendously. Data centers are harvesting huge loads of data because of various new services available recently like virtualization and cloud computing. To make the network infrastructure robust we need to test new network devices at such a load conditions. As a result we need reliable testing tools to test the network infrastructure to find out its weak spots. Currently many organizations rely on commercial hardware based traffic generators like Spirent and Ixia to generate traffic at line rates to test the performance of various prototypes in their labs. These tools are really useful and good in generating various traffic models on a customized hardware with the data speed of line rates. But they have their own limitations as they are quite costly and are not open to experiment with new algorithms or traffic models.
Apart from Hardware based traffic generators we do have software based traffic generators with a GUI, open source, easy to add support for new traffic models and protocols. They are used to generate traffic to test basic features and they are limited in traffic generation rates as well as in performance levels.

II. LIMITATIONS FOR A SOFTWARE BASED TRAFFIC GENERATOR

A software based traffic generator has limitations both on the type of hardware we use as well as on the latency the operating system create for transmitting a packet.

Whenever an user-space application calls send() it actually triggers a number of processing activities. During each send() the operating system must switch context to kernel mode, access the Socket, copy the data to the kernel application buffers, perform protocol related activities, and finally, place the completed packet in the transmit queue. Except for copying data, all processing is proportional to the number of send() calls (rather than to the number of bytes transmitted), so send() function call is a major contributor to per-packet overhead.

As an example, a system call (depending on operating system design and CPU) can require anywhere from a few hundred nanoseconds to several microseconds. System Call overhead can account for as much as 29% of CPU cycles - and Socket related functions, more than 30% as shown in Fig-1.

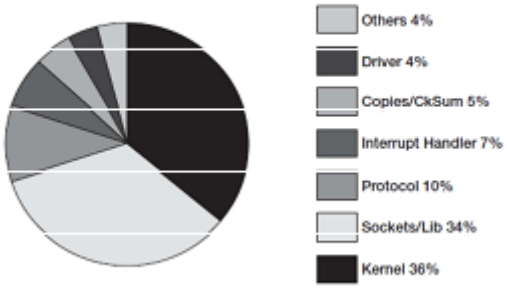

Figure-1: CPU utilization by various process in operating system.

III. EXISTING SYSTEM DESIGN

Let us see various stages through which a packet flows from the user-space until it is transmitted out from the hardware.

For user-space to kernel-space Transmission PF_PACKET is the most widely used socket by most of the software based traffic generators.

Whenever an user-space application calls send (), the operating system must switch context to kernel mode, access the Socket, copy the data to the kernel application buffers. Once packet reaches the kernel space SKB structure (struct sk_buff in Linux) is formed, which acts as an abstract form for all other devices connected can understand the structure.

The SKB is created based on the availability of the memory in transmit buffers of the socket on which we are transmitting the packet. So we need to increase the transmit buffer sizes of the socket to be able to hold the huge burst of traffic we are trying to generate. Now the SKB goes through Linux traffic control (TC), which is purely based on the algorithms being used to control the traffic congestion. This impact adversely on the precision of the transmission time.

Once SKB reaches device driver the whole SKB is removed off and generic packet is taken for the transmission over hardware. This involves multiple copy operations as we have to extract the packet from the SKB structure and transmitted to hardware [9].

IV. BOTTLENECKS IN EXISTING SYSTEM DESIGN

PF_PACKET has many limitations in generating traffic at line rate as mentioned below:

- It uses copy from user function, commonly used to transfer the packet payload to the kernel, and whose overhead is higher than that of a normal memcpy performed to a memory-mapped region [2].
- Packets transmitted via PF_PACKET socket are not directly conveyed to the NIC device driver but go through a series of mechanisms which brings several additional overheads. In particular, packets are also sent to registered sniffers, and therefore toward other open sockets.
- PF_PACKET socket has not been designed exclusively for high rate packet transmission, this results in a severe performance penalty, especially in a multi-core scenario, where several sockets are in-use for parallel transmission (i.e. one socket per thread).
- Linux traffic control (TC), traffic congestion control algorithms forces the packet transmission to be performed asynchronously by a different context, impacting adversely on the precision of the transmission time.

Hardware performs various basic checks on the packet and if successful transmits it and generates an interrupt to let the device driver free the memory allotted for the packet. In high speed packet generation it would be a huge bottleneck as there are so many interrupts to be processed as we are trying to transmit packets at line rate.

As we discussed above there are many bottlenecks which keep on adding for a packet to be transmitted. So there are many things to be considered in designing a new solution to remove the latency created by some of the factors mentioned above.

V. PROPOSED SYSTEM DESIGN

Currently a simplified modular architecture is created where the packet creation is still done in the user-space by the respective software traffic generators. In order to compensate the latency caused by user-space to kernel-space communication a new socket family is created called PF_ZERO, which adds advantage for future extensions without any modifications to kernel default functionality. Traffic generators can just create a socket of type PF_ZERO and start sending the packets without any complexity of knowing the internal implementation [5].

A. Basic Architecture

PF_ZERO internally maintains a memory Mapped circular ring buffers for sharing the data with the kernel module [5]. As shown in below Fig-2

PF_ZERO socket has following components.

i) A memory mapped circular queue for payload and meta-data.
ii) A pool of pre-allocated socket buffers.

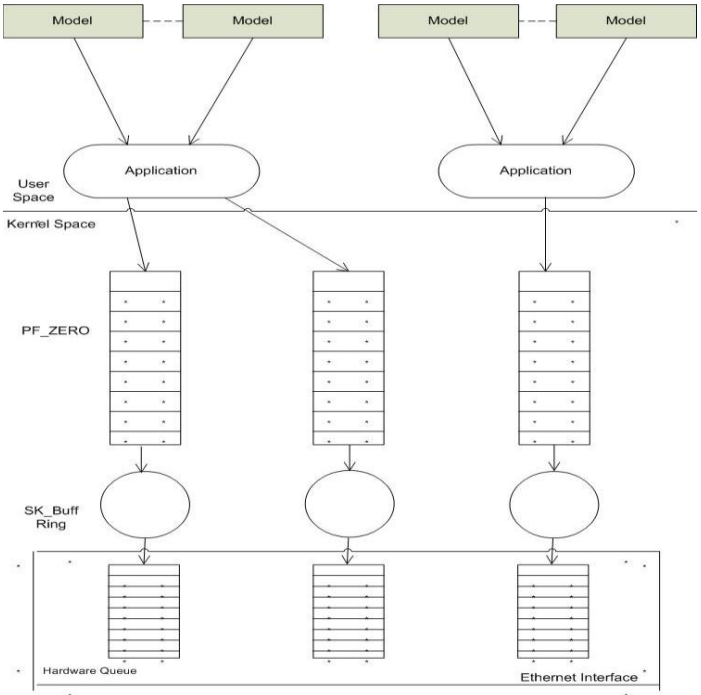

Figure-2: PF_ZERO Socket Architecture.

i) With the first component memory mapped queue the latency of various system calls is removed, which is involved in transmitting the packet from user-space to kernel-space by Using the memory map mmap() feature already available in Linux kernel. A circular buffer is created with predefined number of slots in it. The SPSC (Single Producer Single Consumer) queue is the communication channel between the application and the socket. The queue consists of a memory mapped area and two indices, specifying the last read position and last written position. It avoids the performance cost of a system call and provides a wait-free mechanism for data sharing.
Sample implemented code snippet:

```
sockFd = socket (PF_ZERO, SOCK_RAW, htons(ETH_P_ALL);
……
BufferPtr = mmap(NULL, BufSize,PROT_READ|PROT_WRITE,MAP_SHARED, sockFd, 0);
slotId = &BufferPtr->slotId;
while(TRUE) {
/* Loop until a packet arrives */
if(BufferPtr->slot[slotId].isFull) {
readPacket(BufferPtr->slot [slotId]);
BufferPtr->slot [slotId].isFull = 0;
slotId = (slotId + 1) % BufSize;
}
Else{
writePacket(Bufferptr->slot[slotId]);
BufferPtr->slot [slotId].isFull =1;
slotId = (slotId + 1) % Bufsize;
}}
```

ii) The second component of the socket to be implemented is a ring of pre-allocated socket buffers which are pre-allocated and reused as a result adding performance gains by saving processing cycles for skb creation at run time has the packets are stored in skb ring buffers which are preallocated, instead of creating at runtime. So these ring buffers reduce the overhead involved in creating skb for each packet at run time, by storing packets in pre-allocated socket buffers. To make the device driver not to free the SKB, number of users count can also be increased [5].

It is a try to create solid framework for traffic generation with a new socket family PF_ZERO whose purpose would be for traffic generation unlike PF_PACKET. As a result a new techniques can be added to existing implementation as a kernel module to support new features in future.

VI. EXPERIMENTAL RESULTS

The following packet sizes 64, 128, 512, 1024, 1280, 1500 bytes are used for transmission and comparison of PF_ZERO is done over pktgen, a kernel based packet generator [6].
As shown in below fig-3, the transmission rate for varying packet sizes using linux based traffic generator pktgen has lesser peak.

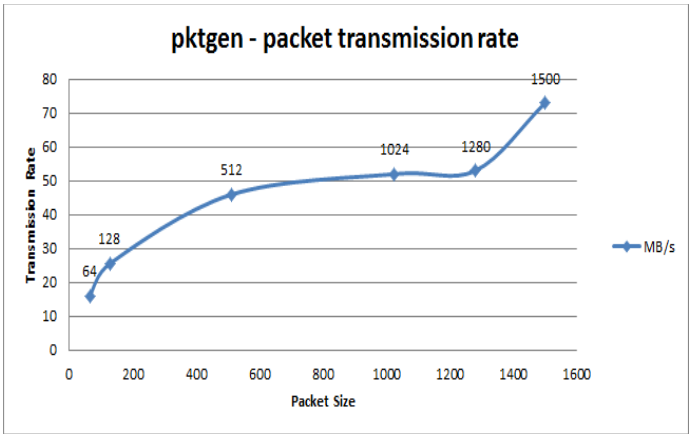

Figure-3: pktgen packet transmission rate

As shown in fig-4 which demos the transmission rate using PF_ZERO which has slight edge over transmission rate over pktgen. PF_ZERO has an advantage of generating wide variety of packets when compared to pktgen using socket type PF_PACKET. PF_ZERO is able to support new protocols types and has the ability to transmit packets from a pcap file, which makes PF_ZERO more feasible to use.

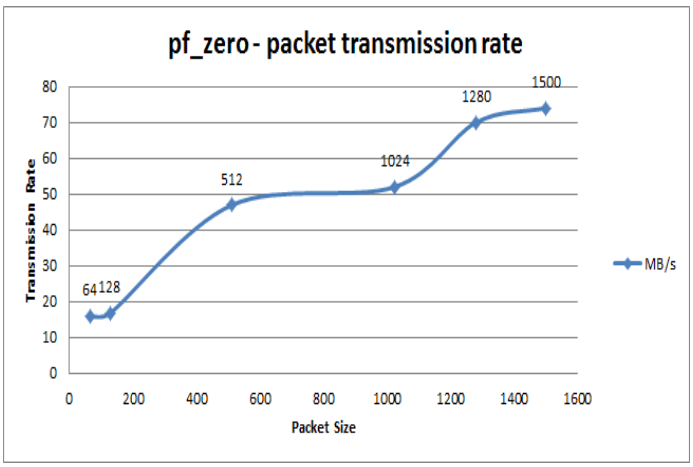

Figure-4: PF_ZERO packet transmission rate.

Even though transmission rates don't make any huge difference cpu utilisation numbers show how much less cpu is needed by PF_ZERO when compared with PF-PACKET implemented pktgen. For all the packet sizes PF_ZERO hasn't crossed cpu utilization rates more than 10% as shown in fig-5 and fig-6.

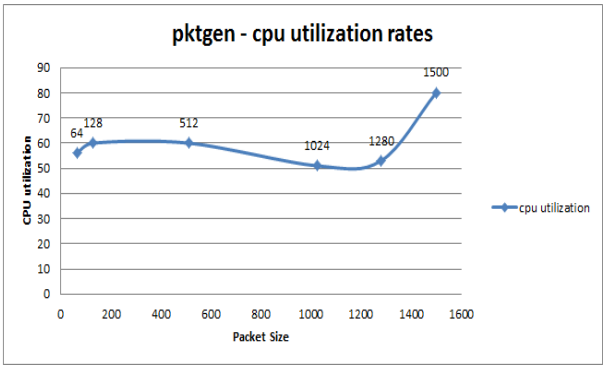

Figure-5: pktgen cpu utilization.

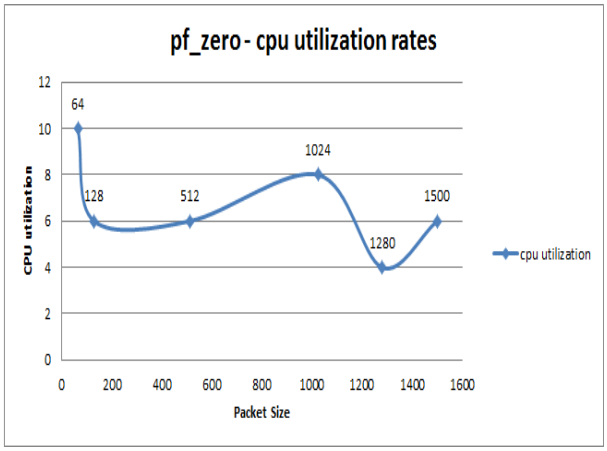

Figure-6: PF_ZERO cpu utilization.

As per the above experimental results even though PF_ZERO was not able to outperform pktgen using PF_PACKET implementation in transmission rates it is able to outperform it in the area of CPU utilization.

VII. CONCLUSION

Traffic generators play a key role in networking environments to test various scenarios in network topologies. Currently we use hardware based traffic generators like ixia and spirent for even small scenarios. Because of their high cost it is not feasible to be bought by everyone who wants to work on networking protocols [7].

An Alternate option is using software based traffic generators on an x86 based desktop. Because of their low cost a desktop can be modified to actually act as a traffic generator with reasonable traffic rates. But even with higher processing speeds they can't generate higher packet rates because of latency in traffic generation mechanism.

This paper addresses the above issue and provides a way to generate higher traffic rates when compared to the existing mechanisms. When we are using a pcap file to generate certain type of traffic because of user space to kernel space communication latency the traffic generated rates will be hugely reduced consuming more amount of CPU speed. On a Linux machine we reduce the latency by using a circular buffer mechanism to transmit and process the packets reducing the latency and in return increasing the packet generation rates to line rates for certain packet sizes.

REFERENCES

[1] Writing device drivers in Linux: A brief tutorial by Xavier Calbet.
[2] mmap , man7.org/linux/man-pages/man2/munmap.2.html
[3] Network drivers by Thomas Petazzoni, Free Electrons.
[4] A Historical View of Network Traffic Models - http://www.cse.wustl.edu/~jain/cse567-06/traffic_models2.htm.
[5] Linux Device driver, 3rd edition, Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman.
[6] Pktgen – linux kernel based packet generator, Robert Olsson.
[7] Do You Trust Your Software-Based Traffic Generator - Alessio Botta, Alberto Dainotti, and Antonio Pescapé, University of Napoli Federico II.
[8] Shared memory concepts, www.kohala.com/start/unpv22e/unpv22e.chap12.pdf.
[9] Skbuffs - A tutorial, ri-oa sissa