TCP/IP PROTOCOL ACCELARATION

[1]*Ketaki Mahabaleshwarkar,* [2]*Neha Mundada,* [3]*Amruta Chavan,* [4]*Amruta Panage*
Department Computer
Pune Institute of Computer Technology, Pune, India

***Abstract*—TCP/UDP/IP and Ethernet have become undisputed leaders of network communication since last decade. TCP/IP stacks are usually implemented in operating system software and packets are handled by the main (host) processor due to which protocol processing of incoming and outgoing network traffic consumes processor cycle. Usually in the application of data transfer, there are several copies of data from user space, kernel space and then NIC which are one of the most CPU hungry operations and hog the CPU upto 80-85%, add latency, consume memory bus bandwidth, and require host processor (CPU) intervention. Zero-copy describes computer operations in which the CPU does not perform the task of copying data from one memory area to another. Using this concept, good data transfer speed can be achieved. If an application has the capability to directly build over the wire data, without kernel intervention, multiple copies could be avoided. The data transfer speed can be increased if kernel is bypassed. The kernel can be bypassed if the application itself can do the functions done by kernel. This can be achieved by providing a separate functionality in the application that performs the task of appending header and checksum. Thus the packet generated by this can directly be sent over the wire. Also every Ethernet card has multiple tx rings amongst which many are not used in most cases. Some of these unused tx rings can be exported to some special purpose application to reduce the copy overhead.**

Keywords: Ethernet, kernel bypass, NIC, TCP/IP protocol, TOE, tx-rings, zero-copy.

I. INTRODUCTION

TCP/IP protocol is the leader of network communication. It can be used to establish connections between different types of computers and servers.This type of interoperability is one of the main advantages of TCP/IP. It is an industry standard, open protocol and is not controlled by one institute. It operates independently of the operating system and includes support for a number of routing protocols. It enables internetworking between organizations. TCP/IP has a scalable, client/server architecture.

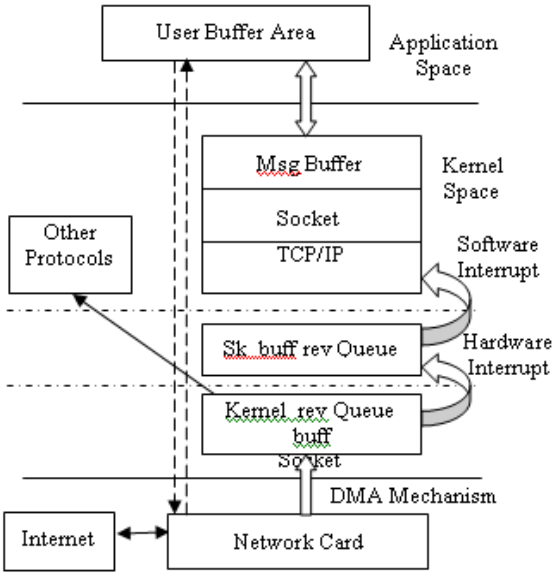

Figure. 1 Traditional TCP/IP model

But despite these advantages there are some loopholes in working of TCP/IP protocol. The TCP/IP stacks are usually implemented in operating system software and packets are handled by the main (host) processor. As a result, protocol processing of incoming and outgoing network traffic consumes processor cycles—cycles that could otherwise be used for business and other productivity applications. As network speed moves beyond 1 gigabit per second (Gb/s) and larger amounts of data are transmitted, processors are burdened by TCP/IP protocol processing and data movement. The burden of protocol stack processing is compounded by a finite amount of memory bus bandwidth. Incoming network data consumes the memory bus bandwidth because each data packet must be transferred in and out of memory several times, received data is written to the device driver buffer, copied into an operating system (OS) buffer, and then copied into application memory space. These copy operations add latency, consume memory bus bandwidth, and require host processor (CPU) intervention.

The TCP/IP protocol overhead associated with 1 Gb of Ethernet traffic can increase system processor utilization by 20 to 30 percent. Today's Ethernet networks, which use TCP/IP operations, commonly operate at 100 megabits per second (Mb/s) and 1 gigabit per second

978-1-4577-1583-9/12/ $26.00 © 2012 IEEE

(Gb/s). Next-generation speeds will increase to 10 Gb/s. Customer migration to 10-Gb Ethernet will be tempered by the input/output (I/O) processing burden that TCP/IP operations place on servers because in traditional systems, the kernel processes messages, causing many times of data copy and a lot of content transforming and at the end resulting in the high delay between network point-to-points. The traditional message process has become the bottleneck of the performance of whole system, as it needs to copy data two times at least, one is from network device to operating system memory, and another one is from system memory to user application space; and also it needs the user sending system calls to the operating system. According to Intel reports, the cost of data copy and its related operations has occupied 69% the cost of whole system. Thereby, removing the useless data copy is a critical process [8].

II. RELATED WORK

Inherent processor overhead and constrained memory bandwidth are performance obstacles for networks that use TCP.

For Ethernet, the use of a TCP/IP offload engine (TOE) and RDMA can diminish these obstacles. However, the TOE is primarily a hardware solution that specifically takes responsibility of TCP/IP operations, while RDMA is a protocol solution that operates at the upper layers of the network communication stack. Consequently, TOEs and RDMA can work together: a TOE can provide localized connectivity with a device while RDMA enhances the data throughput with a more efficient protocol[3]. RDMA technology was developed to move data from the memory of one computer directly into the memory of another computer with minimal involvement from their processors. The RDMA protocol includes information that allows a system to place transferred data directly into its final memory destination without additional or interim data copies. For InfiniBand, RDMA operations provide an even greater performance benefit since InfiniBand architecture was designed with RDMA as a core capability[3].

The hardware solution suggested is called TCP/IP offload engine (TOE). In traditional system architecture, NIC just simply receives data from the host machine and then passes them to network; but an intelligent NIC has programmable processor and memory, a network interface adapter (NIC) with a TOE assumes TCP/IP processing duties, freeing the host processor for other tasks. The capability of a TOE is defined by its hardware design, the OS programming interface, and the application being run[5]. The main disadvantage of TOE is that, the NIC needs to be changed each time the operating system is changed or upgraded. Changing NIC each time is not feasible. Also NIC with a TOE is not economical[7].

The solutions suggested mainly use the concept of zero copy. The basic idea of Zero-Copy is: during grouped data transmission from network device to user application space, by reducing the number of data copies and system calls to realize the zero involvement of CPU, and totally remove CPU workloads in this processing at the end. The main technologies used to implement Zero-Copy are DMA data transmission technology and memory mapping technology [2].

We can summarize these two trends as follows:

1. by using a message processing system to implement message transforming mechanism that walks around the kernel, such as the Zero-Copy technology
2. by trying the best to use the programmable NIC capabilities and bringing into play of hardware transmission speed, such TOE.

III. OUR SOLUTION

Through our project, we propose to implement a new method based on the concept of zero copy i.e kernel bypass. There are many articles on Zero-Copy, but the characteristics of this article is that we mainly concern about its scalability, adaptability, and configurability in Linux and our implementation can work with software based TCP/IP in transferring simple packets[1].

An open source video streaming application would be modified. This application would be given the functionality of packetizing the data itself. The task of packetizing in current architecture is done by kernel. This task mainly includes prepending headers and appending checksum. So our modifications would make the application capable of prepending headers and appending checksum to the packets. Thus the application would be capable of producing data that can be directly sent over wire. Because of our implementation the kernel would be bypassed, no kernel copy would be created. This would reduce the redundancy and also kernel would be free from the task of packetizing and could do some other important task.

The current Network interface cards (NIC) have been designed to support high speed networks, but because of disadvantages of TCP/IP processing they are not used to their extent. The NIC available now-a-days have multiple transmit (tx) rings. Tx ring is an array of buffers which is used to store packets during transmission and reception. Only a few of these rings are used during transmission. So in our solution we propose to use the unused tx rings as well. The application would request access to these unused tx rings. According to the availability, the unused tx rings would be exported to the application. The application will then place the data packets directly in these rings. A timer would be added to synchronize a tx-rx

mechanism for sending out packets enqueued in the ring.

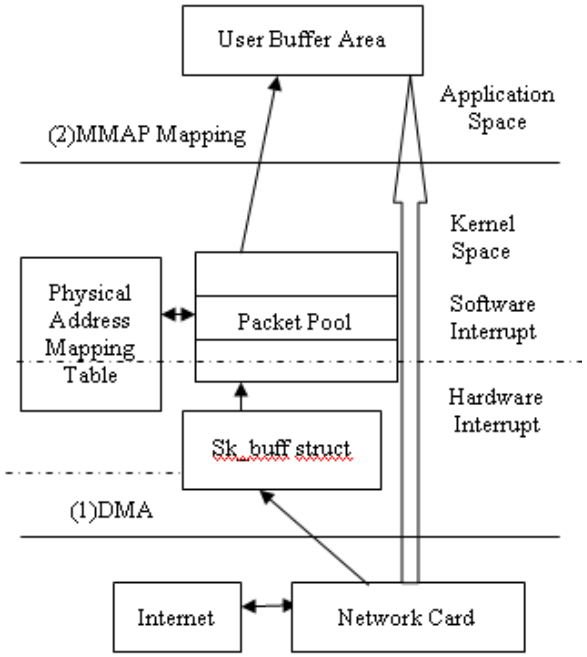

Figure 2. Zero-copy model

There could be some security threats involved in exporting tx-rings but we plan to export only designated transmit ring buffer area to applications. This will reduce potential security loopholes. While this method is in not as safe as usual networking functions, the benefits will be worthwhile using it in controlled environments.

The Structural Design and Analysis of Zero-Copy Model

In Linux system, the normal network data packets transmitting procedure is illustrated as below:

1. User application processes the data packet.
2. User application copies the data packet into user application space by system calls.
3. Software interrupt: program distributes data packets in terms of protocol.
4. Hardware interrupt: program moves data to transmit queue.
5. Network card sends hardware interrupt to host machine.
6. Network card transmits a data packet via to host machine DMA transferred.

In traditional model, when application layer wants to obtain message data, it needs to go through two buffers and the normal TCP/IP protocol stack. Inside, the software interrupt is responsible to receive the message from the first of the receiving queue, and then copy them to msgbuff; at the end application layer reads the message data to user application space by system calls. But in platform with Zero-Copy, it by-passes the protocol stack; when network card is doing DMA operation, it directly transmits the message data to user application space.

Our Zero-Copy implementation contains two steps (here we use data packet transmission as an example):

The first step is to implement DMA data transmission. This step is very closely related to hardware and its device driver. The solution is first to pre-allocate sk_buff (structure) of DMA, and then let the network card receive and store data packets into sk_buff; and finally map the sk_buff into user application space by procedure MMAP.

The second step is the address mapping. Linux system provides a technology which can directly map image file and data file into address space of a process. The content of the file can directly map into virtual address space of a process, this is the memory mapping technology. In the kernel, it allocates sequential space as packet pool, and then it can map this space into user application space by the system procedure MMAP. In user application space, it can allocate sequential space as packet pool. The link table (physical address mapping table) is stored in kernel, and is used to implement the user application space to physical address mapping, and then the packet address of descriptor received by network card can be read directly from this table.

The Implementation of Zero-Copy Idea

Now in the TCP/IP network the main communication is interrupt, memory copy and protocol process. Some experiments show that the cost of data copy and its evocable cost (e.g interrupt, checksum etc.) has occupied 69% of whole cost[13]. In detail, we mainly intend to modify the network device driver source code snull.c, by calling a number of procedures to allocate memories in kernel space, and by assigning them with some value for user applications accessing them.

For memory allocation and its data structure, we can define the maximum allocation page as 512, so the size of each buffer is 1500 bytes. Then adding variable declaration and struct declaration, by get_free_pages procedure to get the number of PAGES page address in kernel space, data can be transferred into user application space address. The members of buffer structure could be width, length, write pointer and read pointer. The unit of buffer in buffer area, its size is based on a group in real Ethernet, it is 1500 bytes. The size of unsigned int is 4 bytes, the rest space is 1496 bytes.

Two procedures are also needed, the buffer deletion procedure: void del_mem() and the initialization buffer procedure: void init_mem().

For accessing buffer space, two procedures are used here: the buffer writing procedure and the buffer reading procedure. The buffer writing procedure parameters may include the content address and the length of content to be written, respectively. The procedure looks up the first buffer with length 0, writes its content and returns its sequential order in buffer space; if lookup reaches to the end of the buffer space, then the lookup goes back to the beginning of buffer space to restart looking up again, unless the whole buffer space has been looked up and the destination buffer location is returned. If the buffer space is full, then returns 0. When reading buffer, another procedure is used to transfer logical address to physical address of the kernel, and then the PROC read procedure is called.

The programming of test program can follow the policies below:

1. Detection whether the buffer contains valid data. The method is to test whether the current packet length is 0.
2. Obtaining the data from buffer. The method is to write the buffer indicator to certain location, then the application retrieves the read location via mapping.
3. Looking up the buffer area to find the buffer with valid data, when the buffer is found, repeat the steps 1 and 2.
4. Using the MMAP procedure to do memory mapping, and to allow application to access the kernel space directly.
5. After read the buffer, set the buffer to WRITE, and free the buffer space

IV. ADVANTAGES

The use of zero copy results in reduction in data redundancy. The CPU time required for packet processing is saved, and the kernel can be used for other productive priority work. The designated unused tx rings in NIC are directly exported leading to reduction in copy overhead. The latency in transmission is reduced because there is no kernel intervention.

V. CONCLUSION

The TCP/IP protocol can be modified using concept of zero copy to support 10Gb/s speed. In this paper, based on the analysis of the traditional message communication mechanism, we propose to improve the current communication mechanism. By implementation of Zero-Copy technology in Linux system, we intend to reduce redundancy, latency and increase the bandwidth. Meantime, by mapping amount of user application space to kernel DMA space, by modifying the network card driver interface to use application buffer directly, we can reduce the message communication path, avoid the cost of memory copy and also drop the CPU workload. Consequently we aim to save a lot of CPU time for application to do any complex computing, and resolved the bottleneck of whole system. This is very useful and has a realistic meaning to high capacity network

REFERENCES

[1]. Design and Implementation of Zero-Copy for Linux Liu Tianhua, Zhu Hongfeng, Liu Jie and Zhou Chuansheng Shenyang Normal University, China

[2]. TCP Implementation in Linux: A Brief Tutorial Helali Bhuiyan, Mark McGinley, Tao Li, Malathi Veeraraghavan University of Virginia

[3]. The Performance Analysis of Linux Networking – Packet Recieving W. Wu, M. Crawford, Fermilab MS-368, Batavia, IL 60510, USA

[4]. Zero copy OS bypass NIC driven Gigabit Ethernet Message Passing Piyush Shivam Computer/Information Science, The Ohio State University

[5]. National Semiconductor, DP8390D/NS32490D NIC Network Interface Controller,July 1995

[6]. The Case for RDMA, Jim Pinkerton, RDMA Consortium 5/29/2002

[7]. Preparing for 2004－2005 Networking Ttransitions Gary Gumanow, Carl Wilson, 2003

[8]. Miguel Rio, Mathieu Goutelle, Tom Kelly, Richard Hughes-Martin-Flatin, andYee-Ting Li, "A Map of the Networking Code in Linux Kernel 2.4.20", March 2004.

[9]. Linux Device Drivers 3 rd Edition by Jonathan Corbet, Greg Kroah Hartman, Alessandro Rubini

[10]. The Linux Kernel Module Programming Guide by Peter Jay Salzman, Michael Burian, Ori Pomerantz

[11]. Ling Li, Liu Haipeng, Research of the Network Software Latency on Linux OS