Performance Enhancement of Network Devices with Multi-Core Processors

Nhat-Phuong Tran, Sugwon Hong*, Myungho Lee, Seung-Jae Lee†
Dept. of Computer Science and Engineering, †Dept. of Electrical Engineering
Myongji University, 38-2 San Namdong, Cheo-In Gu
Yongin, Kyung Ki Do, Korea 449-728
swhong@mju.ac.kr

Abstract— In network based applications, packet capture is the main area that attracts many researchers in developing traffic monitoring systems. Along with the packet capture, many other functions such as security are incorporated in network applications. Specialized hardware and software have been developed and used in order to meet the real-time performance requirements for these functions. Recently, with the prevalence of multi-core processors, researchers are deploying multi-core processor based parallel processing approach to the multi-function network applications. However, parallelizing multiple operations of a network device in an integrated way is difficult because of asynchronous characteristics of coordinating these functions. In this paper, we propose a pipelined parallel execution approach using the producer-consumer model applicable to an environment where a stream of packets passes through two successive processes, each of which performs some tasks on packets. We also implement a packet capture process and an encryption process inside the parallel model, and show the effectiveness of our approach.

Keywords-network device, multi-core, parallel programming

I. INTRODUCTION

In the latest network devices, multiple functions are performed such as packet capture, security, and other high level functions. These functions are typically all harnessed in a single box. For capturing packets, especially high-rate arriving packets, a specialized hardware such as a network processor incorporated in a monitoring card has been traditionally adapted in developing network devices. For security functions, processing a heavy traffic load for enterprise servers is an intimidating task. To overcome the limitation of pure software implementation, some dedicated hardware such as FPGA-based co-processors, hardware accelerators, Graphic Processing Unit (GPU), among others have been developed and used for security functions. Employing dedicated hardware for packet capture, security, and other high level functions respectively can meet the real-time processing requirements imposed on these functions. However, they make the cost of network devices expensive.

Recently, incorporating multiple processor cores on a single chip, or multi-core processor, has become a mainstream microprocessor design trend [1] since mid-2000. As a Chip Multi-Processor (CMP), it can execute multiple software threads on a single chip at the same time. Thus it provides a larger capacity of computations performed per chip for a given time interval (or throughput) [2]. All of the CPU vendors including Intel, AMD, IBM, Oracle/Sun, among others have introduced multi-core processors in the market. Furthermore, the multi-core design is also adopted in embedded processors such as ARM11 MPCore (4-core processor) lately as well.

The advent of multi-core processors ushers in opportunities for many applications to have increased performance. It also encourages software developers to actively utilize parallel programming languages and tools. One of the potential application areas on which these technologies can have profound impact is the development of network devices and servers. Deploying multi-core processor based parallel processing for combined tasks such as packet capture and security functions can have significant impact in this area. However, integrating these essential functions on a single multi-core processor is not easy because of the asynchronous characteristics of coordinating the functions. Due to this difficulty most previous researches have focused on parallel implementation of a single function separately, such as parallel packet capture [3, 4, 5] and parallel data encryption [14]. In this paper we propose a pipelined parallel execution approach to implement the essential network functions, packet capture and security, in an integrated way on a multi-core processor platform. Our approach shows high packet capture rate while processing a security function using a multi-core processor.

The rest of the paper is organized as follows: section II gives an overview of the architecture of multi-core processors. In section III, we explain the issues of packet capture and show the performance of recent packet capturing methods. In section IV we propose a pipelined parallel model for our application consisting of packet capture and security. In section V, we show performance results for our proposed method, compared with the case of the serial mode. Section VI concludes the paper.

II. MULTI-CORE PROCESSOR ARCHITECTURE

Recently, microprocessor designers have been considering many design choices to efficiently utilize the ever increasing effective silicon area with the increase of transistor density. Instead of employing a complicated processor pipeline on a chip with an emphasis on improving single thread's performance, incorporating multiple processor cores on a single chip (or multi-core Processor) has become a main stream microprocessor design trend. As a Chip Multi-Processor (CMP), it can execute multiple software threads on a single chip at the same time. Thus a

* Corresponding author

978-0-7695-4477-9/11 $26.00 © 2011 IEEE
DOI 10.1109/CSE.2011.179

multi-core processor provides a larger capacity of computations performed per chip for a given time interval (or throughput). All the CPU vendors including Intel, AMD, IBM, Oracle/Sun, among others have introduced multi-core processors in the market. The multi-core design is also adopted in embedded systems such as ARM11 MPcore (quad-core) processor based systems introduced lately.

In addition to the CMP based multi-core design, some designs go one step further to incorporate Simultaneous MultiThreading (SMT) or similar technologies on a processor core. Examples are Intel Nehalem and UltraSPARC T2/T3 microprocessor from Oracle/Sun. (Figure 1) shows the architecture of an advanced multi-core processor. On each processor chip, there are *N*-processor cores, with each core having its own level-1 on-chip cache. The *N*-cores share a larger level-2 cache on or off the processor chip. Each core also has *M* hardware threads performing SMT or similar features. Thus it supports two levels of parallelism. For example, the UltraSPARC T2 from Sun includes 8 cores on a chip, with each core supporting 8 hardware threads. In total, 64(= 8 x 8) threads can execute on a chip at the same time. Each core has 8KB private data cache. The level-2 unified cache is 4MB in size

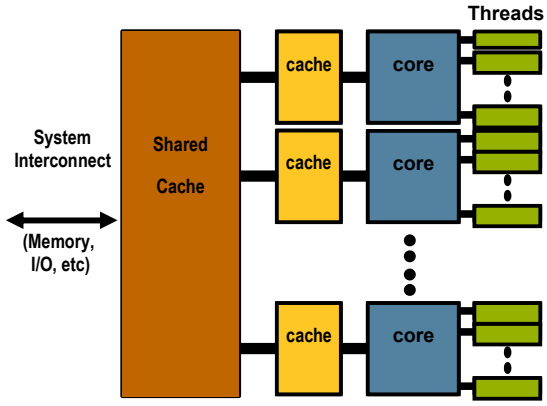

Figure 1. Architecture of an advanced multi-core processors

Although multi-core processors promise to deliver higher chip-level throughput performance than the traditional single-core processors, it is not quite straightforward to exploit its full performance potential. Resources on the multi-core processors such as cache(s), cache/memory bus, functional units, etc., are shared among the cores on the same chip. Software processes or threads running on the cores of the same processor chip compete for the shared resources, which can cause conflicts and hurt performance. Thus exploiting the full performance potential of multi-core processors is a challenging task.

III. PACKET CAPTURE

The fundamental function which network devices are required to do is to capture all incoming packets without any loss if possible. The common packet capture techniques are the socket calls and popular programming library called libpcap. The PF_PACKET socket family allows an application to send and receive packets dealing directly with the network card driver, thus avoiding the usual protocol stack-handling. That is, any packet sent through the socket will be directly passed to the network interface, and any packet received through the interface will be directly passed to the application. Libpcap which has been adopted by many network tools provides platform-independent access to the underlying packet capture facility.

Performance of capturing packets is affected by the kernel livelock in which the operating system is interrupted to handle incoming packets. OS has to spend most of its time processing interrupts in heavy traffic situation, resulting in poor performance [6]. Many efforts have been tried to improve the performance of packet capture and transmission, eliminating Kernel livelock while processing interrupts [7, 8, 9].

One of the main problems in the typical packet capturing techniques based on PF_PACKET and libpcap is to copy packets from the network interface card to the kernel space buffer and then again copy packets from the kernel space to user space application buffer. Memory copy needs CPU cycles and memory bandwidths. The zero copy technique eliminates the additional memory copy by the kernel copying packets directly from the network card to the user space buffer. PF_RING is the new type of socket which enables the kernel to utilize the zero copy method [8]. Figure 2 shows improvement of packet capturing rate by PF_RING comparing with the traditional methods, PF_PACKET and libpcap, where the receiver has Intel Core 2 Duo 2.2GHz CPU and 2GB RAM running the Centos 5.5 operating system.

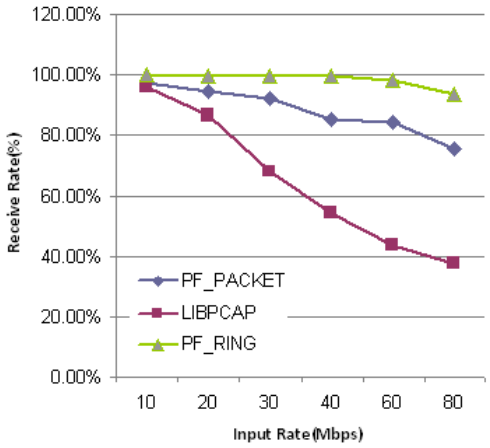

Figure 2. Packet capture rate of different methods

Modern network interface cards (NIC) provide multiple RX/TX queues and balance packet flow over these queues using hardware-based facility [11]. If we use multi-core system, it is expected that multiple threads running on multiple cores can process packets in multiple queues in NIC in parallel mode, and achieve drastic performance improvement. But the current kernel does not support multiple interfaces between application threads and NIC queues, but rather packet polling in the kernel fetches

packets into the user threads through a single interface, as shown in figure 3.

One proposal to cope with this problem is to use a new NIC driver model called threaded NAPI (TNAPI) with the PF_RING buffer [3]. The TNAPI spawns one polling thread for each RX queue. Each polling thread fetches packets in the corresponding RX queue, and passes them to the PF_RING buffer which is under care of the application thread in the user space. The paper shows that this method brings out significant performance improvement while the use of double RX queue and threads on dual cores performs worse than a single queue and thread implementation without TNAPI [3].

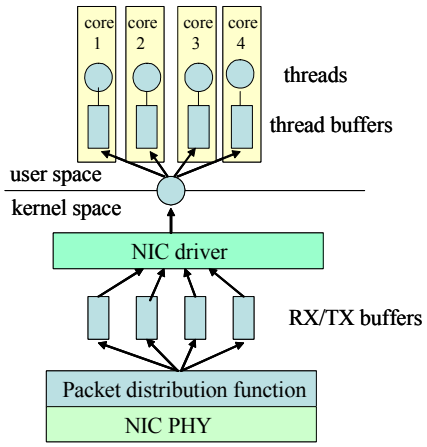

Figure 3. packet capture architecture in multi-core system

IV. PIPELINED PARALLEL EXECUTION OF PACKET CAPTURE AND SECURITY

Passive packet capture is the integral task of network traffic monitoring tools. But packet capture is just the beginning step leading to more sophisticated functions in other network devices. Most network devices in the future are expected to have multiple functions including communication function and maybe security function, all harnessed in a single box. Figure 4 shows the basic functional blocks inside the typical network device. First the device should capture all incoming packets flowing over network. At the next step the cryptographic computation should be applied to most, if not all, incoming packets to guarantee secure communication. These functions could be considered as the preprocessing step for high-level application processes.

Traditionally for capturing packets, especially high-rate arriving packets, the specialized hardware such as network processors in the monitoring cards is adapted in developing network devices. Implementing the security function under heavy traffic load is intimidating task to enterprise servers. To overcome the limitation of pure software implementation, some dedicated hardware, which may be FPGA-based co-processors or hardware accelerators or graphic processor (GPU), are used only for security functions.

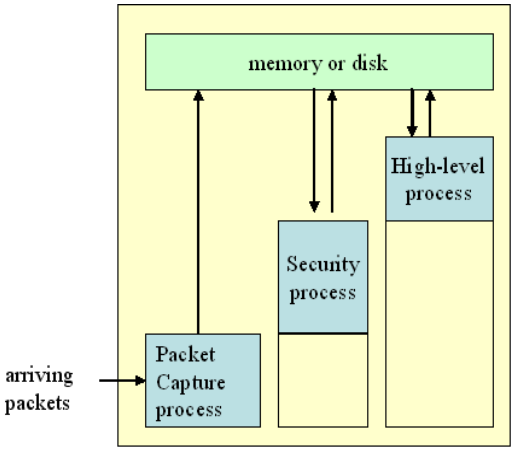

Figure 4. basic functions in a network device

One possible and less expensive alternative for developing the network devices in the future is to use the general purpose multi-core processor, avoiding any specialized hardware, and consequently reducing development costs significantly. However, to program for multiple interrelated application processes to execute in parallel is not an easy problem to solve. Some processes are not independent from the perspective of sharing data they have to use and synchronizing their tasks. For example, the process which performs encryption function is working on the packet which was captured and moved into the user space buffer by the packet capture process. The task of the former process is triggered when packet copy is completed in shared memory by the latter process.

The proper parallel programming model for this purpose might be the producer-consumer model [12]. In this model a stream of data pass through successive processes, each of which performs some tasks on data. A proceeding process can be considered as a producer of the data stream which the following process consumes. Production of new data by a producer process triggers the action of a task by a consumer process. Thus, a chain of producer processes and consumer processes can operate as a pipeline. One of drawbacks of this approach would be load balancing due to heavily coarse granularity. Because workload of each process is different and varied, some process will take longer time to produce or consume data, causing unbalanced CPU utilization between cores.

The following code snippets explain our pipelined approach. In work(), two threads (1, 2) execute asynchronously while performing its own functions (read_input and signal_read for 1; wait_read and process_data for 2). Thread 1 performs packet capture using read_input. Two threads synchronizes and data is passed from 1 to 2 by singnal_read and wait_read. Thread 2, after receiving data from 1, performs the security function by executing process_data.

```
work()
begin
        int sock;
        int i;
        init arrays
        sock = init socket

        while (1)
           make parallel areas
             1st parallel area
             begin
                for (i = 1,i<4, i++)
                  read_input(sock, i);
                  signal_read(i);
                end for
             end
             2nd parallel area
             begin
                for (i = 1,i<4, i++)
                  wait_read(i);
                  process_data(i);
                        end for
                end
            end parallel areas
        end while
end
```

Fugure 5. Code snippet for pipelined execution (Part 1)

```
read_input(sock, i)
begin
        byte *buffer, newdata;
        int k, newlen;
        allocate memory to buffer
        while (k < ELEMENTS)
                buffer = receive packet
                newlen = normalization(buffer,
                          newdata)
                if (i%TASKS = 1)
                        move packet to array1
                else if(i mod TASK =0)
                        move packet to array2
                end if
        end while
end
```

Fugure 6. Code snippet for pipelined execution (Part 2)

```
signal_read(int i)
begin
        readflag[i mod TASKS] = 1
end
```

Fugure 7. Code snippet for pipelined execution (Part 3)

```
wait_read(int i)
begin
        refresh readflag
        while (readflag[i mod TASKS] = 0)
                refresh readflag
        end while
        return
end
```

Fugure 8. Code snippet for pipelined execution (Part 4)

```
process_data(int i)
begin
        int k;
        if (readflag[i mod TASKS] = 1) {
                if (i mod TASKS = 0)
                parallel for-loop here
                for (k = 0, k < ELEMENTS, k++)
                    encrypt(array2[k].data,
                            array2[k].len)
                end for
                else if (i mod TASKS = 1)
                parallel for-loop here
                for (k = 0, k < ELEMENTS, k++)
                    encrypt(array1[k].data,
                            array1[k].len)
                end for
                end if
        end if
        readflag[i mod TASKS] = 0;
end
```

Fugure 9. Code snippet for pipelined execution (Part 5)

One of serious issues to implement this parallel model is management of access to shared data. As shown in the multi-core processor architecture in figure 1, when data is retrieved, data traverses from shared memory to dedicated cache to each core. Most multi-core processor have more than two-level caches (Lelve-1 and Level-2 caches) that are dedicated to or shared between cores depending on which level the cache is. The cache hit rate which implicates data locality on memory affects performance significantly. When a process has to use data which are scattered widely on memory, it is likely to have low cache hit rate, which means to retrieve data from memory more frequently. One example is a packet-processing application that is performing TCP reassembly on a large number of TCP flows, which means to access a large number of data over many memory locations [5]. The result in the article [5] shows how much the number of TCP connections affects cache hit rate and throughput.

In addition to avoiding cache-trashing to minimize the number of packet copies between buffers in two processes is critical in exploiting parallel implementation. The occurrence of packet copy triggers another task in the pipelined model, which imposes the cost of synchronization.

V. EXPERIMENTAL RESULTS

In the experiment below, we execute a packet capture process and an encryption process in parallel based on the producer-consumer model explained in the previous section. We use the SEED algorithm as an encryption method which is a 128-bit symmetric key block cipher developed by Korea Information Security Agency in 1998, and has been since adopted by most of the security systems in Korea [13]. The parallel programming is implemented using OpenMP.

In this implementation, to avoid excessive accesses to buffers and decrease the cost of synchronization, double buffers are used to store the packets in PF_PACKET and libpcap methods. At first encryption process must wait until one buffer is full. While the encryption process is accessing

one buffer, arriving packets are being stored in another buffer. When one buffer is full, a coordinating function triggers the encryption process. When one buffer is full and encryption of packets in another buffer has not completed yet, captured packets are stored in temporary buffer until encryption is completed. There will be cost for copying from the temporary buffer to any buffer and for waiting buffer full at the initial time. However, it can decrease the cost needed for synchronization when two CPU access the same buffer more frequently. Before storing packets into buffers, the length of packets is normalized to be aligned with multiple of 16 bytes.

In this experiment we use the same system as a receiver which is used for the packet capture experiment in figure 2 (Intel Core 2 Duo 2.2GHz CPU and 2GB RAM running the Centos 5.5 operating system). We generate 10,000,000 packets with the size of 512 bytes and arriving rate of 60Mbps. We obtain packet capture rates for three different methods (LIBPCAL, PF_PACKET, PF_RING). Then we measure the packet capture rates when a single core performs packet capture followed by packet encryption. TABLE I below summarizes the two results. When a single core performs both functions, capture rates drop noticeably ranging 13.0~14.7%.

TABLE I. PACKET CAPTURE RATE OF THREE DIFFERENT METHODS

Methods	Packet Capture Rates	
	Only packet capture	Packet capture + encryption
PF_PACKET	97.21%	83.69%
LIBPCAP	87.49%	74.61%
PF_RING	96.90%	84.31%

By using the pipeline execution as described in Section III, packet capture and security functions are separately performed on different cores. TABLE II below compares the results of 2-core execution results vs. 1-core execution of two functions. Figure 10 also shows the performance comparisons. The comparisons shows that significant portion of the performance drop (packet capture rate) shown in TABLE I due to executing two functions on the same core got recovered by executing them in parallel on two separate cores. We also admit that this gain can be furthered, considering that there is still room of improvement to reduce the synchronization cost mentioned in the previous section.

TABLE II. PACKET CAPTURE RATE OF THREE DIFFERENT METHODS

Methods	Packet Capture Rates		
	Only packet capture	Packet capture + encryption (1-core)	Packet capture + encryption (2-cores)
PF_PACKET	97.21%	83.69%	94%
LIBPCAP	87.49%	74.61%	82.59%
PF_RING	96.90%	84.31%	95.36%

VI. CONCLUSION

In this paper, we proposed a pipeline parallel execution model for processing packet capture and security functions commonly used in network based applications concurrently on a multi-core processor. Implementing multiple such functions for a network application on a general-purpose multi-core processor requires careful design of a parallel execution model and consideration to reduce the parallelization overheads such as synchronization cost caused by packet copy and cache-trashing. Experimental results show the effectiveness of our parallel model in balanced concurrent execution of the two functions. This furthermore results in enhanced packet capture rates by distributing the two functions on different cores. Further work is planned to implement possible reduction of the synchronization cost associated with the asynchronous communication while executing the two functions.

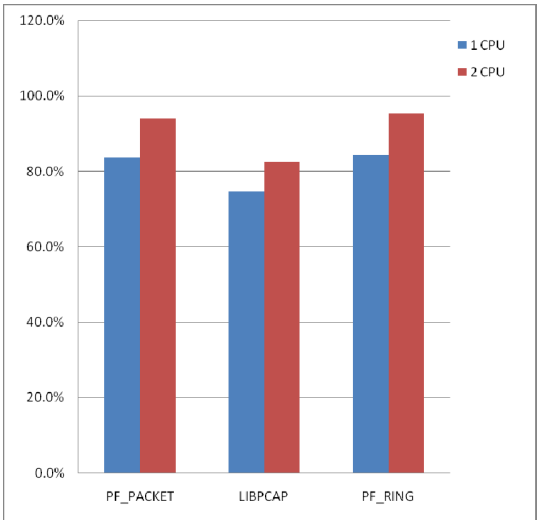

Figure 10. Packet capture rate with encryption process

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the ministry of Education, Science, and Technology (Grant No: 2009-0089793).

REFERENCES

[1] L. Spracklen and S. Abraham, Chip MultiThreading: Opportunities and Challenges, 11th International Symposium on High-Performance Computer Architecture (HPCA-11), pp 248-252, 2005.

[2] Y. Li, D. Brooks, Z. Hu, K. Shadron, "Performance, Energy, and Thermal Considerations for SMT and CMP Architectures," 11th International Symposium on High-Performance Computer Architecture, 2005.endon, 1892, pp.68–73.

[3] F. Fusco and L. Deri, "High-Speed Network Traffic Analysis with Commodity Multi-Core System," http://svn.ntop.org/imc2010.pdf.

[4] F. Yang, Y. Dou, Z. Lei, and J. Yu, "Performance anaylsis of high rate packet capture on multiprocessor platform," Front. Elecr. Electron. Eng. China 2010, Vol. 5, No 1, pp36-42.

[5] E. Verplanke, "Understand packet processing with multi-core processors," EE Times-India, April 2007.

[6] J. C. Mogul and K. K. Ramakrishnanl "Eliminating receive livelock in an interrupted-driven kernel," ACM Trans. On Computer System, Vol. 15, No 3, pp217-252, 1997.

[7] L. Rizzo, "Device Polling support for FreeBSD," the EuroBSDCon 2001.

[8] L. Deri, "Improving passive packet packet capture: Beyond device polling," the 4th Int. System Administration and Network Engineering Conference 2004.

[9] L. Deri, "nCap: Wire-speed packet capture and transmission," the IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services 2005.

[10] F. Schneider, J. Wallerich, and A. Feldmann, "Packet Capture in 10-Gigabit Ethernet Environments Using Contemporary Commodity Hardware," LNCS 4427, pp207-217, 2007.

[11] Intel, Accelerating high-speed networking with intel i/o acceleration technology, White Paper 2006.

[12] M. Quinn, Parallel Programming in C with MPI and OpenMP, McGraw Hill, 2004.

[13] H.J.Lee, S.J. Lee, J.H.Yoon, D.H.Cheon, J.I.Lee, "The SEED Encryption Algorithm," IETF RFC 4269, December 2005.

[14] A.D. Biagio, A. Barenghi, G. Agosta, G. Pelosi, "Design of a Parallel AES for Graphics Hardware using the CUDA framework", Proceedins of the 2009 IEEE International Symposium on Parallel & Distributed Processing, May 2009.