제30회 한국정보처리학회 추계학술발표대회 논문집 제15권 제2호 (2008. 11)

개선된 PF_RING을 이용한 고성능 패킷 캡쳐

단조위*, 김용수*
*경원대학교 IT대학
e-mail:chaoyi_duan@hotmail.com, kimys@kyungwon.ac.kr

Improved PF_RING for High Performance Packet Capture

Chao Yi Duan*, Yong Soo Kim*
*IT College, Kyungwon University

Abstract

The packet capturing becomes a bottleneck in the network intrusion detection and monitoring system as the network performance developing. Many approaches, zero copy, interrupt coalescing and NAPI which attempt to improve packet capturing performance of Linux, are inefficient. PF_RING is a new type of network socket that dramatically improves the packet capture speed, but not perfect. This paper proposes some solutions which can improve the memory utilization and save some data copy time based on the commodity network adapters rather than on the commercial network adapters.

1. Introduction

On the past several years, people tried to improve the performance of packet capture on host PC not only on the software side but also on the hardware side. There are many packet capture tools, such as Tcpdump[1], Ethereal[2], Snort[3], nProbe[4], PF_RING[5] and nCap[6] and so on. All of them are based on a popular programming library called libpcap[7] which provides a high level interface to packet capture. This library is very useful but not perfect because of many limitations to capture packets on Linux.

Besides, there are many approaches such as zero copy[8], interrupt coalescing[9] and NAPI[10] and so on which are not efficient to solve packet capturing performance of Linux. PF_RING is a better way.

PF_RING is a high speed packet capture library that turns a commodity PC into an efficient and cheap network measurement box suitable for both packet and active traffic analysis and manipulation. Moreover, PF_RING opens totally new markets as it enables the creation of efficient application such as traffic balancers or packet filters in a matter of lines of codes.

The PF_RING creates a new type of socket (PF_RING) optimized for packet capture that is based on a circular buffer (ring buffer) where incoming packets are copied.

The advantages of a ring buffer located into the socket are manifold, including:[11]

(1) Packets are not queued into kernel network data structures.

(2) The mmap primitive allows userspace applications to access the circular buffer with no overhead due to system calls as in the case of socket calls.

(3) Even with kernel that does not support device polling, under strong traffic conditions the system is usable. This is because the time necessary to handle the interrupt is very limited compared to normal packet handling.

(4) Implementing packet sampling is very simple and effective, as sampled packets do not need to be passed to upper layers then discarded as it happens with conventional libpcap-based applications.

(5) Multiple applications can open several PF_RING socket simultaneously without cross interference (e.g. the slowest application does not slow the fastest application down).

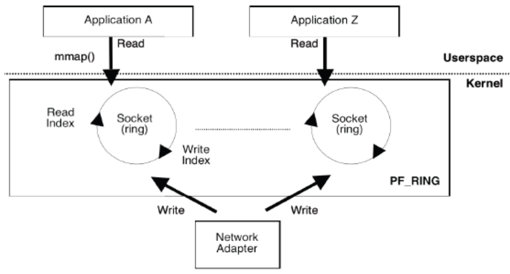

Figure 1. PF_RING Socket Architecture

Memory and time always are the very important issues on high performance software. For PF_RING, the ring buffer will take too much memory if there are so many PF_RING sockets created. And the data copy time will be the bottleneck when mass packets are captured on the network adapter especially the gigabit network adapter. In this paper, we propose some solutions to reduce the ring buffer size and save the data copy time when moving the packets into the ring buffer.

2. Motivation

In the PF_RING socket, the ring buffer is allocated when the socket is created, and released when the socket is deactivated. Different sockets will have a private ring buffer. In other words, every user application has its own PF_RING socket. If there are quite a number of user applications, there will be quite a few ring buffers which take up too much memory. Therefore, it's a good idea to reduce the memory allocated. But the each socket one ring buffer solution can avoid the interaction between the different applications because the slowest application does not slow the fastest application down. According to the ring buffer element number can be set by the application, the only way is to reduce the ring buffer element memory requirement.

In the PF_RING socket, the ring buffer element consists of list_head structure and sock structure. We don't parse the packet in the kernel and the sock structure is very large because it includes too much information which is not useful besides the socket buffer structure (sk_buffer). In this paper, we propose a new structure called pf_buff structure which is very simple but very useful instead of sock structure.

Whenever a packet is received from the adapter (usually via DMA, direct memory access), the driver passes the packet to upper layers (on Linux this is implemented by the netif_receive_skb() and netif_rx() functions depending whether polling is enabled or not). In case the PF_RING socket, every incoming packet is copied into the socket ring buffer or discarded if necessary. (e.g. in case of sampling when the specified sample rate has not been satisfied). In this copy operation, we do nothing else but copying packet data into ring buffer because the packet data has to be transferred to the application without being modified. The copy operation wastes much time and much memory, this paper proposes a function called skb_clone() which just copy the socket buffer pointer rather than the packet data instead of skb_copy() function.[12]

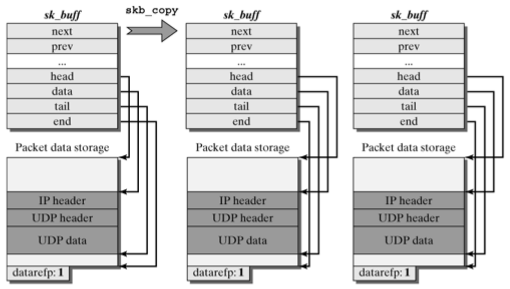

Figure 2. Copying socket buffer

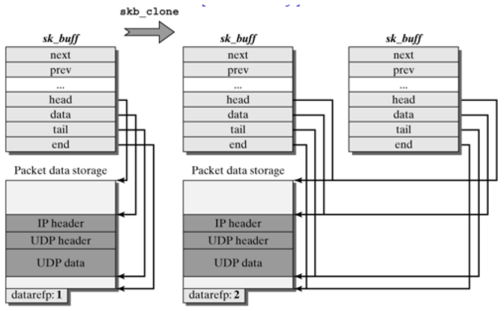

Figure 3. Cloning socket buffer

Figure 2 shows the situation before and after skb_copy() function is called and Figure 3 shows the situation before and after skb_clone() function is called. From these two figures, we can clearly see that in skb_copy() function we have to allocate another memory to copy the packet data storage, but in skb_clone() function, we don't need to do that.

3. Our proposals and implementation

In order to reduce the ring buffer element memory requirement, we propose a new structure called pf_buff as following:

```
/* add a new structure instead of sock */
struct pf_buff {
  u_short   sk_family;
  struct    sk_buff *skb;
  void      *sk_protinfo;
  void      (* sk_destruct) (struct sock *sk);
};
```

Figure 4. The pf_buff Structure

In this new structure, the variable sk_family shows the socket family, in PF_RING socket, it is PF_RING; the structure pointer skb points to the socket buffer which contains the receiving packet; the pointer sk_protinfo indicates the protocol operations; the function pointer sk_destruct() points to the function which will destruct the sock structure. In this structure, we discard so many items which are not used in the PF_RING socket.

```
//The original ring element structure
struct ring_element {
  struct list_head  list;
  struct sock       *sk;
};
```

Figure 5. The Original Ring Buffer Element Stucture

```
//The modified ring element structure
struct ring_element {
  struct list_head  list;
  struct pf_buff    *sk;
};
```

Figure 6. The Modified Ring Buffer Element Structure

Figure 5 shows the original ring buffer element structure and Figure 6 shows the modified ring buffer element structure. In our experiment, we use pf_buff structure to replace the sock structure. In Figure 4, we can see that the pf_buff structure is very small which can save much memory every socket.

Skb_copy() function creates a copy of the socket buffer, copying both the sk_buff structure and the packet data. It spends too much time and memory to call alloc_skb() function which allocates memory for a socket buffer structure and the corresponding packet memory. Skb_clone() function also creates a new socket buffer; however, it allocates only one new socket buffer structure, and no second space for packet data. The pointers of the original sk_buff structure and of the new structure point to the same packet data space. This allows us to prevent the time-intensive copying of a complete packet data space when a packet is to be copied into the ring buffer. The memory containing packet data is not released before the variable datarefp contains a value of one (i.e., when there is only one reference to the packet data space left). So in this paper, we make skb_clone() replace of the skb_copy() to save the CPU time and memory when the incoming packets are copied into the ring buffer.

4. Experiments

In our experiments, we make a Linux server installed Fedora operating system as our experimental platform. And we compare performances of PF_RING with our proposed study against four criteria: (i) the ring buffer memory requirement (ii) the latency to copy socket buffer.

4.1 Configuration

Our experimental platform consists of two end machines, which are connected by Ethernet. One is packet generator which feeds 64Byte, 512Byte, 1500Byte UDP/IP packets to the other machine. The other one is installed PF_RING and improved PF_RING on which we did our experiments. The configuration of the receiver machine is in the following table.

Table 1. The Platform Configuration

Parameters	Configuration
OS Release Version	Fedora Core release 6
OS Kernel Version	Linux 2.6.25.3
Network Adapter	100baseTx-FD
CPU Property	Intel(R) Xeon(TM) CPU 3.00GHz
CPU Number	4
Memory Size	2GB

4.2 The Ring Buffer Memory Requirement

Table 2. Ring Buffer Element Size Evaluation

Ring Buffer Element	PF_RING	Improved PF_RING
Sock structure	448 Bytes	--
Pf_buff structure	--	12 Bytes
List_head structure	8 Bytes	8 Bytes

As the Table 2 shows, the change is very big after pf_buff structure instead of sock structure, in the original PF_RING socket, every ring buffer element should be allocated 448 + 8 = 456 Bytes; in the improved PF_RING socket, every ring buffer element is just allocated 12 + 8 = 20 Bytes. The default ring buffer length is 4096, therefore, our improved PF_RING socket can save 4096 * (456 - 20) = 1785856 Bytes, about 1.7 Mbytes every socket.

Table 3. Socket Buffer Evaluation

Packet Size	PF_RING	Improved PF_RING
64 Bytes	228 Bytes	164 Bytes
512 Bytes	676 Bytes	164 Bytes
1500 Bytes	1664 Bytes	164 Bytes

In the Table 3, it shows memory is allocated between PF_RING and improved PF_RING when socket buffer (sk_buff structure) is copied into the ring buffer based on different packet sizes. Because of the skb_clone() function, the improved PF_RING socket doesn't depend on the packet size. From this table, the improved PF_RING can save more memory as the packet size increasing.

4.3 The Latency

Table 4. The UDP Packet Processing Latency

Packet Size	PF_RING	Improved PF_RING
64 Bytes	9.9µs	9.8µs
512 Bytes	42.2µs	41.6µs
1500 Bytes	119.0µs	117.6µs

As the Table 4 shows, the packet processing latency in improved PF_RING is shorter than in original PF_RING. That's because in original PF_RING, the skb_copy() function has to call alloc_skb() function to allocate the memory for the packet data; the skb_clone() function in the improved PF_RING does not.

5. Future works

Although PF_RING is a very effective tool for capturing packets on a Gbit network, it still leads to packet loss because of the ring buffer overflow. In future, we can solve the overflow problem to make no packet be lost. We also can do a study of the features that can be implemented with respect to packet transmission in order to have a complete send/receive architecture.

References

[1] The Tcpdump Group, tcpdump, http://www.tcpdump.org/

[2] G. Combs, Ethereal, http://www.ethereal.com/

[3] M. Roesch "Snort-Lightweight Intrusion Detection for Networks" Proceedings of Usenix Lisa '99 Conference, http://www.snort.org/

[4] L. Deri "nProbe: an Open Source NetFlow Probe for Gigabit Networks" Proceedings of Terena TNC 2003, Zagreb, May 2003

[5] Amitava Biswas, Purnendu Sinha "A High Performance Packet Capturing Support For Alarm Management System" 17th IASTED International conference on Parallel and Distributed Computing and Systems, Phoenix, AZ, Nov 2005

[6] L. Deri "nCap: wire-speed packet capture and transmission" Proceedings of the End-to-End Monitoring Techniques and Services on 2005. Workshop, May 15-April 30, 2005, pp. 47-55

[7] Lawrence Berkeley National Labs, libpcap, Network Research Group, http://www.tcpdump.org/

[8] P. Wang, Z. Liu "Operating system support for high performance networking" a survey. http://www.cs.iupui.edu/~zliu/doc/os_survey.pdf

[9] Wen-Fong Wang, Jun-Yau Wang, Jin-Jie Li "Study on Enhanced Strategies for TCP/IP Offload Engines" Proceedings of the 2005 11th International Conference on Parallel and Distributed Systems, July 2005, pp. 398-404

[10] J.H. Salim, R. Olsson "Beyond softnet" 5th Annual Linux Showcase & Conference, Oakland, CA, 2001, pp. 165-172

[11] Luca Deri "Improving Passive Packet Capture: Beyond Device Polling" SANE 2004, September 2004, pp. 1-12

[12] Klaus Wehrle, Frank Pahlke, Hartmut Ritter, Daniel Muller, Marc Bechler "The Linux Networking Architecture" Pearson Education, 2005