ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.9, September 2007, pp.2271−2282
DOI: 10.1360/jos182271
© 2007 by *Journal of Software*. All rights reserved.

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种可扩展的高效入侵监测平台技术*

杨 武[1+], 方滨兴[2], 云晓春[2]

[1](哈尔滨工程大学 信息安全研究中心,黑龙江 哈尔滨 150001)
[2](哈尔滨工业大学 计算机网络与信息安全技术研究中心,黑龙江 哈尔滨 150001)

Techniques of Building a Scalable, Efficient Intrusion Monitoring Architecture

YANG Wu[1+], FANG Bin-Xing[2], YUN Xiao-Chun[2]

[1](Information Security Research Center, Harbin Engineering University, Harbin 150001, China)
[2](Computer Network and Information Security Technique Research Center, Harbin Institute of Technology, Harbin 150001, China)
+ Corresponding author: Phn: +86-451-82589638, Fax: +86-451-82589638, E-mail: yangwu@isrc.hrbeu.edu.cn, http://isrc.hrbeu.edu.cn

Yang W, Fang BX, Yun XC. Techniques of building a scalable, efficient intrusion monitoring architecture. *Journal of Software*, 2007,18(9):2271−2282. http://www.jos.org.cn/1000-9825/18/2271.htm

Abstract: To perform effective intrusion analysis in higher bandwidth network, this paper studies the data collecting techniques and proposes a scalable efficient intrusion monitoring architecture (SEIMA) for network intrusion detection system (NIDS). In the architecture of SEIMA, scaling network intrusion detection to high network speeds can be achieved using multiple sensors operating in parallel coupled with a suitable load balancing traffic splitter. High-performance data transfer is achieved through asynchronous DMA without OS's intervention by using efficient address translation technique and buffer management mechanism. Multi-rule packet filter based on finite state machine technique is implemented at user layer to eliminate overhead for processing redundant packets. The simulative and actual experiment results indicate that SEIMA is capable of reducing the using rate of CPU while improving the efficiency of data collection in NIDS, so as to save much more system resources for complex data analysis in NIDS. The method of SEIMA is very practical for network security.

Key words: intrusion monitoring; load balance; data collection; address translation; packet filter; security analysis

摘 要: 为了在更高带宽的网络中进行有效的入侵检测分析,研究了入侵检测中的数据获取技术,提出了一种可扩展的高效入侵监测框架 SEIMA(scalable efficient intrusion monitoring architecture).在 SEIMA 结构模型中,通过将高效网络流量负载分割器与多个并行工作的入侵检测传感器相结合,从而可以将入侵检测扩展应用到更高的网络带宽中;通过使用高效地址翻译技术和缓冲区管理机制实现了旁路操作系统的高性能用户级网络报文传输模型,以便提高单传感器的报文处理性能;通过采用有限自动机的方法构建了基于用户层的多规则报文过滤器以消除多余数据包的处理开销.模拟环境和实际环境下的测试结果表明,SEIMA 在提高网络入侵检测系统数据获取效率的同时,能够降低系统 CPU 的利用率,从而可以将更多的系统资源用于更复杂的数据分析过程.

关键词: 入侵检测;负载均衡;数据收集;地址翻译;报文过滤;安全分析

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2002AA142020 (国家高技术研究发展计划(863)); the National Information Security 242 Program of China under Grant No.2005A33 (国家 242 信息安全计划)

Received 2005-01-24; Accepted 2006-03-31

中图法分类号: TP393　　文献标识码: A

1 Introduction

With the rapidly growing connectivity of the Internet, networked computer systems are increasingly playing vital roles in our modern society. While the Internet has brought great benefits to this society, it has also made critical systems vulnerable to malicious attacks. Network security is more and more important. Since a preventive approach such as firewall is not sufficient to provide sufficient security for a computer system, NIDS is introduced as a second line of defense and becomes a research hotspot in the fields of network security.

Generally, NIDS captures and filters network packets by monitoring the key network segment passively, and then uses all kinds of detecting algorithms to find the proof of intrusion from these packets. Once the attacking events are validated, NIDS either breaks the network connections initiated by the intruders by sending the packet with flag of RST (FIN) or cooperates with firewall to stop intrusion from happening actively.

Effective intrusion detection requires significant computational resources: widely deployed systems such as snort[1] need to match packet headers and payloads against tens of header rules and often many hundreds of strings defining attack signatures. This task is much more expensive than the typical header processing performed by packet forwarders and firewalls. With the rapid development of the network techniques, the switched network of 100Mbps and 1000Mbps has gradually replaced the traditional shared LAN of 10Mbps, and the network bandwidth is also increased by many times. The constant increase in network speed and throughput poses new challenges to the real-time processing performance of NIDS. Therefore, performing effective intrusion detection at high network speeds (e.g. 1Gbit/s and beyond) requires further improvement on performance of data collection and data analysis in NIDS.

This paper focuses on the process of data collection in NIDS, presents a scalable efficient intrusion monitoring architecture (SEIMA) for NIDS. In SEIMA, multiple node machines operate in parallel, fed by a suitable traffic splitter element to meet the requirement of different network traffic. A high-performance user-level messaging mechanism (ULMM) and a user-level packet filter (ULPF) are presented and implemented in order to improve the packet capture performance and packet processing efficiency on a single machine. The intrusion monitoring platform designed in this paper is highly applicable and deserves popularizing not only for NIDS in a heavy traffic network but also for other network applications such as firewalls and network charging systems, etc.

2 Related Work

Research on network intrusion detection has been ongoing for many years for producing a number of viable systems. But with the increasing network throughput in recent years, it is becoming a very important problem how to split network traffic for reducing payload on a single machine, improve packet capture performance and packet filter efficiency for eliminating unnecessary packets in high traffic network.

Kruegel, *et al*.[2] presented a parallel cluster intrusion detection model based on traffic partition. This approach adopts a slicing mechanism that divides the overall network traffic into subsets of manageable size and accordingly reduces the traffic load on a single machine.

The traditional endpoint packet capture systems generally use Libpcap (library of packet capture)[3] based on in-kernel TCP/IP protocol stack, but the slow network fabrics and the presence of the OS in the critical path (e.g. the system call overheads, in-kernel protocol implementations, interrupt handling and data copies) are the main bottlenecks on every packet sending and receiving. Therefore, inefficient Libpcap cannot adapt to the environment of heavy traffic network.

Based on the loadable kernel module technique (LKM), Libpacket[4] reduces the system overhead of context switch by saving certain numbers of packets in the allocated kernel buffer and reading multiple packets in a single system call. In essence, the layered structure of user-kernel in Libpacket doesn't remove the kernel from the critical path of data transfer. The main performance bottlenecks are still in existence.

To eliminate main performance bottlenecks during communication completely, zero-copy protocol architectures for cluster systems were presented, including U-Net/MM[5], VIA[6] and VMMC[7]. These architectures adopt the flat structure of user-hardware, fully bypassing in-kernel protocol stack in OS and allowing applications of the direct access to the network interface. The Virtual Interface Architecture (VIA) is connected oriented: each VI instance (VI) is specially connected to another VI and thus can only send to and receive from its connected VI. The U-Net/MM and VMMC architectures integrate a partial virtual address translation look-aside buffer into the network interface (NI) and allow network buffer pages to be pinned and unpinned dynamically, coordinate its operation with the operating system's virtual memory subsystem in case of a TLB miss. This definitely increases the implementation complexity of network card firmware and causes the great overhead because of frequent pinning/unpinning buffer and requesting pages. In addition, VMMC commonly requires customized high-speed interconnection network and NI hardware. Whereas, the NIDS passively monitor the TCP/IP network. So the above communication architectures are not suitable for high-speed network intrusion detection systems.

The traditional packet filter such as BPF[8] is commonly implemented in OS kernel. When the received packets are not interesting to the user application, BPF will dropt these packets so as to save system overhead for copying them from kernel to application. BPF matches packets with multiple filter rules one by one in checking packets, thus BPF's processing efficiency is low when the number of rules is many.

3 Key Techniques in Designing and Implementing SEIMA

At high-speed network monitoring spot, data processing performance of a single machine may reach threshold so as not to meet the need of real-time intrusion analysis. We make use of load balance technique to build a scalable parallel intrusion monitoring architecture, which adopts the computing mode of SPMD to extend or shrink by network traffic. The scalable intrusion monitoring model is shown in Fig.1, which mainly consists of three parts: IDS load balancer, packet transfer module and packet filter module.

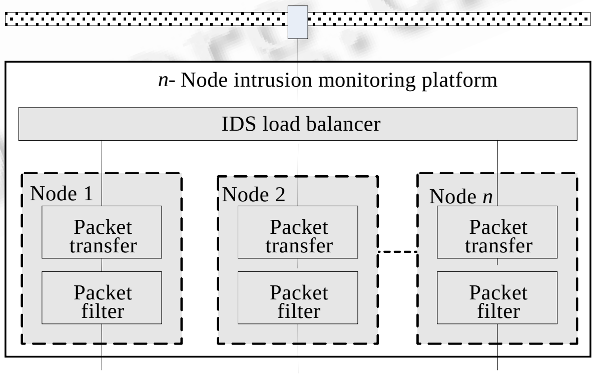

Fig.1 Scalable intrusion monitoring model

The IDS load balancer is a custom-built device that connects to high-speed network through optical splitter. To split network data stream entering load balancer into each processing node of parallel cluster monitoring architecture, load balancer needs being configured with Ethernet network interface of 100Mbps/1000Mbps. A user-level messaging mechanism based on zero-copy technique is employed in packet transfer module, in concert with which a user-level multi-rule packet filter is implemented in packet filter module. The number of processing nodes varies with the actual network traffic, one-node machine can also satisfy the requirement of performance in the common network traffic.

3.1 Connection round robin based load balance algorithm

Since load balancer commonly needs the processing of a large number of network packets, complex algorithm will reduce the processing efficiency of load balancer. Thus load balancer adopts a simple efficient load-balancing algorithm for data splitting. The majority of network packets are based on TCP protocol in current heavy traffic backbone network. For network traffic of TCP connection, an ideal load balance algorithm should meet the following requirements: 1) data is almost evenly split into every node to ensure approximate load balance among node machines; 2) bi-directional data of any TCP connection is split into single node to ensure data independence among node machines. So a load balance algorithm based on connection round robin is employed in the load balancer, which makes rational and nearly even data splitting with granularity of connection.

For the TCP protocol, a four-tuple with the form of ⟨source IP, destination IP, source port, destination port⟩ uniquely defines a connection. The connection round robin scheduling algorithm is described in Fig.2.

```
Given: $N$ is the number of node machines;
          $m$ is one-node machine number distributed recently, $m\in[1,N]$;
          $A$ is one-node machine number obtained currently, $A\in[1,N]$;
Initialize $m$=1;
For every packet p of TCP protocol {
     If p is the first packet of a connection (SYN packet)
         the entry address obtained $A=m \bmod N+1$;
         split p into node machine $A$;
            record four-tuple of this new connection and $A$ in HASH table;
            $m=A$;
     Else
            look up HASH table to find entry address $A$;
            split p into node machine $A$;
            if p is the last packet of a connection or connection is overtime
                remove this connection record from HASH table;
}
```

Fig.2 Load balance algorithm based on connection round robin

For other protocol type (e.g. UDP, ICMP), the entry address may be computed by the simple and direct hashing of two-tuple ⟨source IP, destination IP⟩. The detailed formula is as follows:

$$\textit{Destination node number}=(\textit{source}\ \mathrm{IP}\oplus \textit{destination}\ \mathrm{IP}) \bmod N.$$

3.2 Efficient user-level messaging mechanism-ulmm

For improving packet-processing performance of one-node machine and reducing the cost of hardware resource, a zero-copy, bypass-OS user-level messaging mechanism (ULMM) for intrusion detection is presented and implemented. In ULMM, the critical path of data communication is removed from OS kernel, thus messages can be transferred directly to and from the user-space applications by the network interface without any intermediate steps. The performance of message sending and receiving is greatly improved.

The operation mode of passively monitoring in NIDS indicates that both sides of communication are not requested to resend packets, even if there are errors in the received messages by NIDS. In this case, error control is a useless time-consuming operation. In addition, it is unnecessary that ULMM uses the means of flow control to adjust message transfer speed. ULMM eliminates the system overhead of dynamic allocating/releasing buffer and pinning/unpinning buffer by allocating a continuous static user-space buffer and pinning corresponding physical memory pages. Caching the whole virtual-to-physical address table in the kernel space removes the overhead from the necessary address translation operation of the operating system's virtual memory subsystem in case of partial virtual-to-physical address table miss in U-Net/MM.

Message transfer model of ULMM is compared with that in the traditional NIDS, which is shown in Fig.3. Fig.3(b) clearly shows the architecture of ULMM in three gray modules: Kernel Agent (K-agent), New NIC Driver, and User-Level Network Interface (ULNI). The block arrows describe the data flow, while the line arrows describe the control flow. The module of ULNI provides the API library for the application, in which API functions are defined in Table 1. Kernel Agent and New NIC Driver do all the real work to copy the data from NIC memory to user process's memory. Specifically, Kernel Agent obtains the physical addresses of the application's memory range and then creates the buffer ring. This buffer ring holds all the packets copied from NIC waiting to be filtered by the packet filter in Section 3.3. And the New NIC Driver asks Kernel Agent for the physical address table of this buffer ring which is used by asynchronous DMA of NIC and initiates DMA to transfer packets between the application's buffer ring and the on-chip memory of NIC. In comparison with the traditional message transfer model in traditional NIDS, ULMM greatly improves the packet capture performance bypassing the system kernel, which makes packet capture more practical in high-speed network.

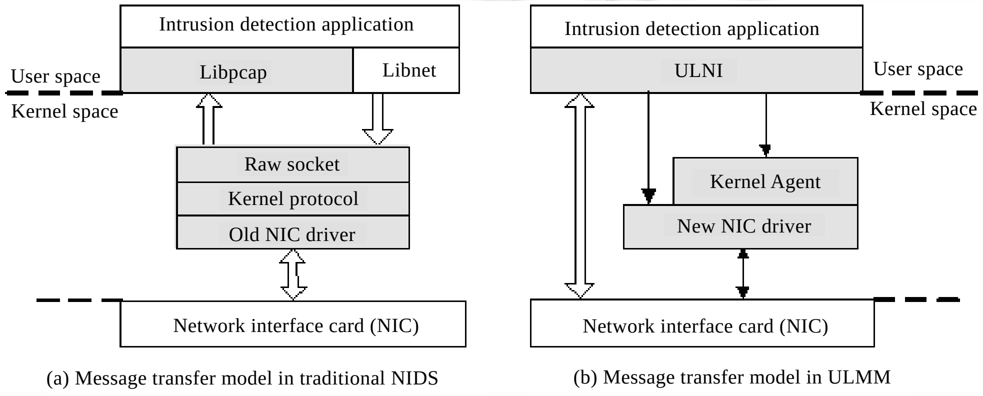
Intrusion detection application
Libpcap
Libnet
User space
Kernel space
Raw socket
Kernel protocol
Old NIC driver
Network interface card (NIC)
(a) Message transfer model in traditional NIDS
Intrusion detection application
ULNI
User space
Kernel space
Kernel Agent
New NIC driver
Network interface card (NIC)
(b) Message transfer model in ULMM

Fig.3 Comparison of two message transfer models

Table 1 API library for ULMM

ULMM_OPEN	Open ULNI and do initialization
ULMM_LOOP	Callback function for application
ULMM_GETDATABLOCK	Read packets from user buffer in ULNI
ULMM_SENDDATABLOCK	Send packets in user buffer in ULNI
ULMM_FREEDATABLOCK	Release used buffer block

3.2.1 Translation mechanism for virtual address in user space

ULMM directly transfers packets between the application's buffer and the on-chip memory of NIC by asynchronous DMA. The use of DMA transfer between host and network interface memory introduces two problems. First, most systems require that every host memory page involved in a DMA transfer be pinned to prevent the operating system from replacing that page. Second, typically the network interface's DMA engine must know the physical address of each page it transfers data to or from. Since the DMA facility accesses the physical memory address space, whereas application uses virtual address space, one main difficulty in designing ULMM is the translation between virtual address of user buffer and physical address accessed by DMA. In addition, if each message transfer involves pin-down and release kernel primitives, message transfer bandwidth will decrease since those primitives are quite expensive. So, how to pin/unpin physical pages is another problem to consider.

The user application allocates a continuous user-space memory as message buffer statically (we call it *UserBuff*) and coordinates with a loadable kernel module (K-agent) to inform it about the starting virtual address (*UserBuffAddr*) and size of user buffer through API library provided by ULMM. Linux kernel currently uses a three level page table for virtual address translation. K-Agent completes translation from virtual address to physical address and pins physical pages by means of related kernel functions operating on the page table. The translated physical addresses are cached in kernel space in the form of the virtual-to-physical address table (*PhyAddressTable*) and *PhyAddressTable* covers all the physical addresses of the user buffer blocks accessed by network interface, which avoids unnecessary address translation operation of the operating system's virtual memory subsystem in case of the partial physical address table miss in U-Net/MM. The method to acquire some virtual address's physical address and pin it is described as follows, in which *page_num* is the pre-computed number of pages, the initial value of *vir_addr* is *UserBuffAddr*.

```
For (i=0; i<page_num; i++, vir_ddr+=PAGE_SIZE)
    {
    /*physical address translation for user buffer*/
    pgd_t*pgd=pgd_offset(current→mm, (unsigned long) UserBuffAddr);
    pmd_t*pmd=pmd_offset(pgd, (unsigned long) vir_ddr);
    pte_t*pte=pte_offset(pmd, (unsigned long) vir_ddr);
    PhyAddressTable[i]=(pte_val(*pte) & PAGE_MASK);
    /*lock the user buffer page*/
    page=pte_page(*pte);
    set_bit (PG_locked, & page→flags);
    atomic_inc(& page→count);
}
```

3.2.2 Message buffer management mechanism supporting multi-thread

High performance user-level communication architecture generally requires a large memory to cache messages, which can achieve a very high throughput. ULMM saves packets in a big buffer allocated in user space statically, buffer management mechanism of which is shown in Fig.4. This figure indicates that the user buffer is divided into sending buffer and receiving buffer that are separately used during packet sending and receiving, which makes ULMM support full duplex communication mode and avoid mutex lock operation.

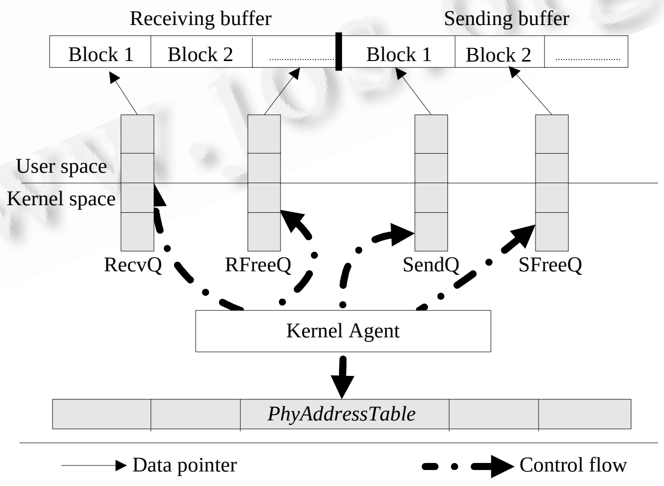

Fig.4 Message buffer management mechanism of ULMM

Every user buffer is also divided into many buffer blocks with size of 2KB, each of which is for saving a network packet frame. For supporting application's multi-thread access to message buffer without data copies on SMP processor so as to improve application's packet processing efficiency, K-Agent allocates four buffer rings to manage the user buffer in kernel space: sending busy ring (SendQ), receiving busy ring (RecvQ), sending free ring (SFreeQ) and receiving free ring (RFreeQ), each of which includes descriptor items for buffer blocks. The item of data block to be sent is put in SendQ ring, the item of the received data block is put in RecvQ ring and the item of the free data block is put in SFreeQ or RFreeQ ring.

Each item structure consists of two fields 〈*index*,*size*〉: 1. *Index* corresponds to block number; 2. *Size* corresponds to size of packet in data block. ULMM maps four kernel rings into user space by virtue of *mmap* function provided by memory mapping mechanism in Linux, so as to make user process and kernel module share buffer management rings.

3.2.3 Efficient packet sending and receiving mechanism

Message transfer process in ULMM is shown in Fig.5. Packet receiving process is ①→②→③→④→⑤→⑥, and a detailed description of which is as follows: When a new packet arrives, new NIC Driver gets the item of a free data block from the head of RFreeQ ring and acquires pinned physical address of this free data block from the physical address table (*PhyAddressTable*) according to the block number (*index*) of the item, and then initializes asynchronous DMA to transfer packets. When DMA transfer is finished, an interrupt is generated. New NIC Driver puts the item of the data block just filled with packet at the tail of RecvQ ring in interrupt handler. When the application needs to process packets, application gets the item of a busy data block with packet from the head of RecvQ ring and reads the packet in this user buffer block in terms of *index* of the obtained item. After application processes the packet, it puts the item of the data block just used at the tail of RFreeQ ring. In a similar manner, the packet sending process is (1)→(2)→(3)→(4)→(5)→(6) in turn.

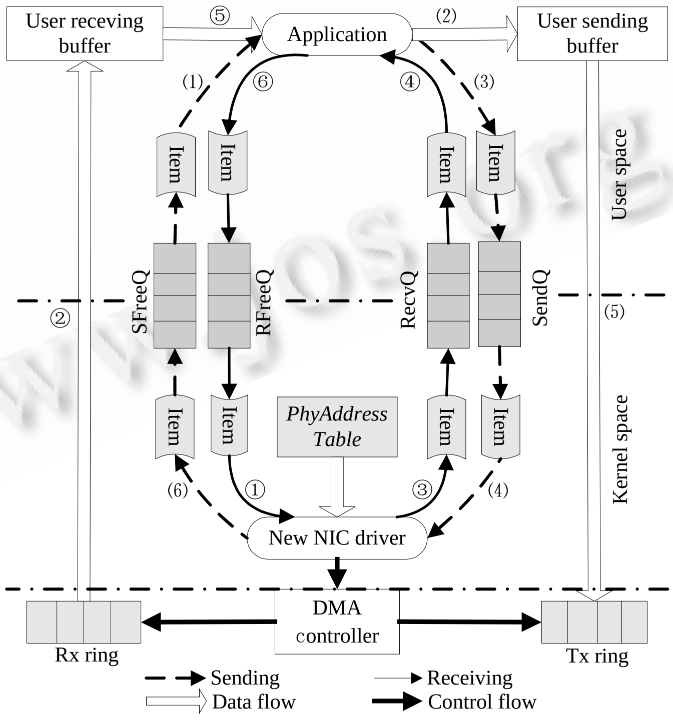

Fig.5 Packet sending and receiving process in ULMM

3.2.4 Simulative testing for ULMM

For high-performance network intrusion detection systems, packet capture is the main consideration. In the following, we focus on testing the packet capture performance of ULMM and compare with those of the existent packet capture models.

Simulative experiment environment is as follows: Two machines are connected with each other by Ethernet in the same local domain. One is special for packet sending (configuration: Router Tester GbE/4 1000Base); the other is special for packet capture (configuration: CPU-PIII1G*2, Memory-2G, Network card-Intel 1000Mbps Ethernet card). The packet capture mechanism is separately Libpcap, Libpacket and ULMM.

We test peak throughputs of Libpcap, Libcapture, Old NIC Driver and ULMM on different packet sizes and the results are shown in Fig. 6. Peak throughput of ULMM increases with the packet size and reaches its threshold at the point of about 1500B because of zero-copy technique that eliminates the memory copy between the kernel and the user applications, which shows that ULMM is a high-performance packet capture library. Peak bandwidth of Libpcap does not vary too much with packet size and reaches peak value of 196.38Mbps at the point of about 512B, this is the result of the traditional kernel protocol used by Libpcap. For each different packet sizes the throughput of ULMM is much greater than that of Libpcap or Libpacket. Fig. 6 also indicates that there is a crossover between the curves of ULMM and Old NIC Driver, and this is because Old NIC Driver uses the limited kernel buffer allocated/released dynamically to cache packets, ULMM employs a big static buffer in the user space for saving packets in order to eliminate the overhead of frequent allocating/releasing buffer, whereas the overhead of Old NIC Driver’s allocating/releasing buffer declines with the gradual increment of the packet size.

Figure 7 shows the results of the average CPU idle rates of Libpcap, Libpacket and ULMM with different packet sizes at peak throughput. The CPU idle rate of ULMM increases with the packet size because the system overheads of interrupt handling and DMA initialization declines continually. The CPU idle rate of Libpcap reaches the lowest value at the point of 256B, and this is the result of the tradeoff between the overhead of hard interrupt handling and that of data memory copy. For each different packet size, the average CPU idle rate of ULMM is much greater than that of Libpcap or Libpacket, which indicates that NIDS will save more CPU cycles for other computing tasks besides the packet capture.

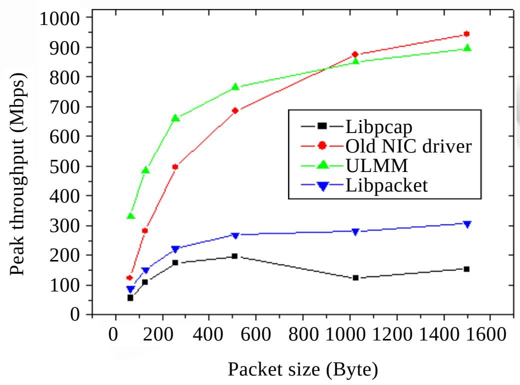

Fig.6 Peak throughput of libpcap, libpacket, old NIC driver and ULMM with different packet sizes

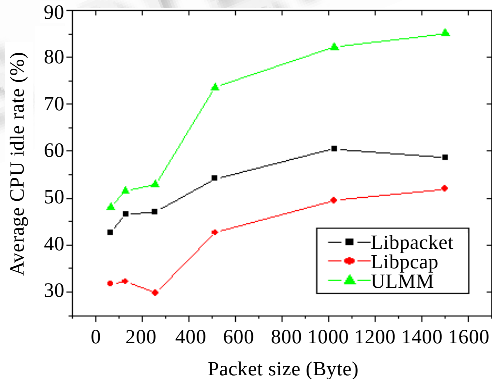

Fig.7 Average CPU idle rates of libpcap, libpacket and ULMM with different packet sizes at peak throughput

3.3 Multi-Rule user-level packet filter-ULPF

The packet filter is a subsystem to reduce the volume of data to be analyzed by the security analysis module by removing non-interesting network packets, and at the same time protects the NIDS itself from hostile attacks such as DOS/DDOS. To keep away from packet dropping, the packet filter simply analyzes some fields of the packet header, such as IP address, TCP/UDP port number and TCP flag. After this filter, the number of the packets left for the detection analysis is largely decreased. Based on the zero-copy user-level messaging mechanism, a multi-rule packet filter-ULPF (user-level packet filter) is implemented at the user layer.

The traditional packet filters, e.g. BPF, are often implemented in the system kernel and drop those non-interesting packets according to the application's requirements. Different from the traditional packet filters, ULPF has to be implemented in the user processes to work with the zero-copy ULMM in Section 3.2.

ULPF uses a rule description language to define fields to be checked in the packet headers, and the action is followed once the packet satisfies a precondition. A precondition is composed of equations and logical operators in ASF-like grammar. The filtering rules are the form of "*packet*|*precondition*→*action*". For example, a rule is defined as follows: *packet*(*p*)|(*p.e_type*=*ETHER_IP*) && (*p.protocol*!=*IP_TCP*) && (*p.protocol*!=*IP_UDP*) && (*p.protocol*!=*IP_ICMP*) && (*p.protocol*!=*IP_IGMP*)→*Drop*, means that Ethernet packet will be dropped if its protocol type of IP layer is unknown.

In practice, the packet filter often uses a larger number of filter rules. To satisfy the requirements of both the performance and multi-rule filter, ULPF is designed for the following goals:

1) Single scan of the packet. That is, a packet can be processed only once and all the rules satisfied by the packet should be found.

2) Changeless time. That is, the system execution time is insensitive to the number of rules. For example, when the number of rules doubles, the matching time is much less than double.

We build the multi-rule packet filter model in ULPF as a DFA (deterministic finite automata). At first, all the filter rules are preprocessed and a DFA is built from all the equations, and then the header fields of the analyzed packet are scanned from the left to the right and go through the DFA. During the scanning, the unrelated fields are skipped as an idea of adaptive pattern matching presented in Ref.[9], which speeds up the packet filter. The algorithm for automata construction is shown in Fig. 8. Function *Build*() is recursive and the entire automata can be established by invoking *Build*(*root*), where the root is associated with an empty matching set and a full candidate set containing all of the specified rules.

We capture packets from the actual network environment with *Tcpdump* tool and save them in

```
void Build (struct node v) {/* v is a node in automaton, extra information are attached
                  to each node: p is the field offset to be inspected, m is the set of already
                  matched rules and c is the set of candidate rules */
   If (v.c is empty)
      return;                          /* if no candidate rule, terminate the procedure */
   v.p=select(v.c);                    /* select the field offset to inspect in the node v */
   buildbranch (v.p);               /* create all the possible branches of node v, each branch
                                         has a edge to it from v, with corresponding value */
   for (each rule r in v.c){
      if (r has test relevant to v.p) {
         if (test for equality) {
            if (r can be matched after this test)
               add r into matched rule set of the branch with corresponding value;
            else
               add r into candidate rule set of the branch with corresponding value;
         }
         if (test for inequality) {
            if (r can be matched after this test)
               add r into matched rule set of the branch with corresponding value;
            else
               add r into candidate rule set of all the branches except the branch
               with corresponding value;
         }
      }
      else
         add r into candidate rule set of all branches;
   }
   for (each branch v′)
      Build (v′);           /* recursively call Build for v′ */
}
```

Fig.8 Algorithm for automatic construction of ULPF

test.tcpdump file. We test packet-filtering time of ULPF and BPF with different number of filter rules on test.tcpdump and the results are shown in Fig.9. The processing time of BPF gradually increases with the increment of the number of filter rules, because BPF checks rules one by one for every packet. ULPF uses an automatic algorithm to build packet filter model, so its processing time is nearly invariant with the number of rules. For each number of filter rules, the processing efficiency of ULPF is much greater than that of BPF.

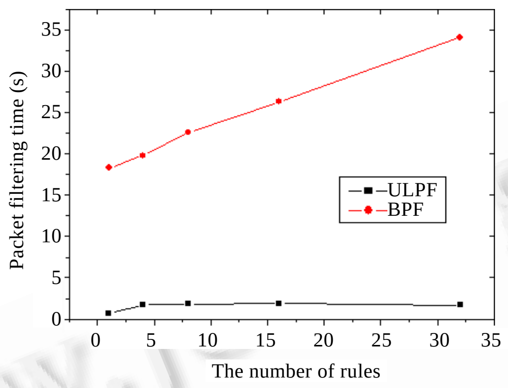

Fig.9 Time of packet filtering with different number of filter rules

4 Testing Results in Actual Environment

1. To validate the scalability of SEIMA, we test the relation between the actual connecting network bandwidth and the minimal number of node machines that is needed. The results are shown in table 2. The traffic load on each node machine is almost even and each node machine seldom loses packets in every group of experiment. From this table, we can include that SEIMA is capable of processing the ever-increasing data volume in a high-speed network at the cost of certain hardware resources.

Table 2 The results of experiments on SEIMA

Actual network bandwidth (Mbps)	612	2 192	4 656
The number of node machine	2	8	16

2. ULMM based on zero copy is the core component in SEIMA, and performance of ULMM decides the cost and practicability of monitoring platform. To further validate the practicability of SEIMA, we test ULMM and compare it with other models in the actual environment. The actual testing environment is described in Fig.10, in which the connecting optical fiber is from the ISP (internet service provider) of China Telecom and the bandwidth is about 1.6Gbps. Three testing machines are of the same configuration: CPU-PIII1G*2, Memory-2G, Network card-Intel 1000Mbps Ethernet card. We separately run ULMM, Libpcap and Libpacket on testing machine 1, testing machine 2 and testing machine 3. The results of experiments are shown in Fig.11 and Fig.12, which indicate that packet capture performance and the CPU idle rate of ULMM are the highest of the three packet capture models in actual environment. So SEIMA

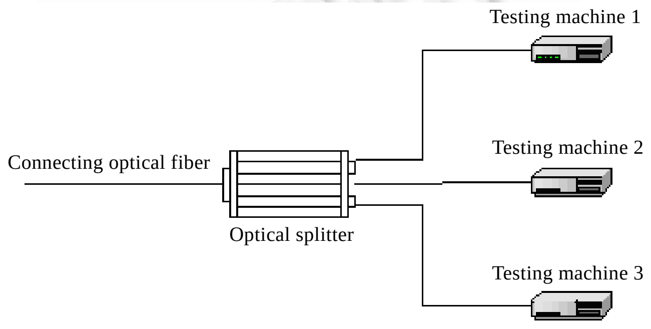

Fig.10 The actual testing environment

which adopts ULMM can save much more system resources for the complex data analysis in NIDS.

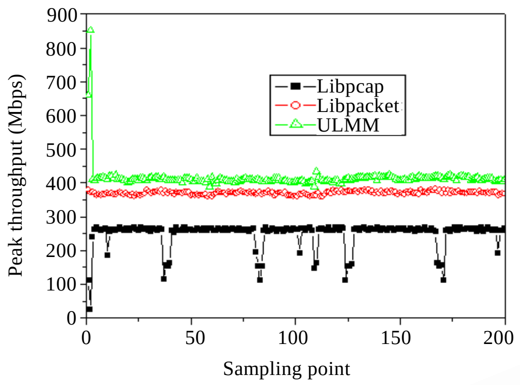

Fig.11 Processing performances of three packet capture models in actual environment

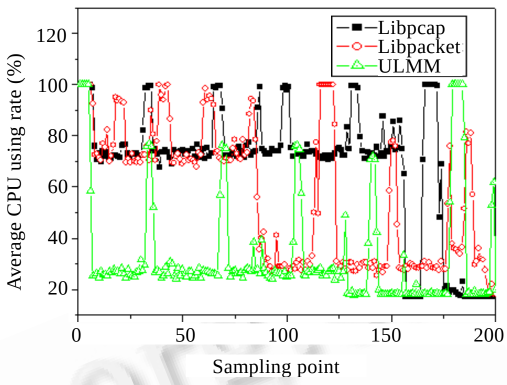

Fig.12 CPU using rates of three packet capture models in actual environment

3. We use snort 2.0 as an intrusion detection experiment system, modify the source code of snort 2.0 and separately use Libpcap and ULMM as its packet capture interface. Thus we separately run snort+Libpcap and snort+ULMM on testing machine 1 and testing machine 2 described in Fig.10. The results of experiments are shown in Fig.13 and Fig.14, which can be drawn that ULMM can indeed greatly improve packet processing performance and the CPU idle rate of the intrusion detection system.

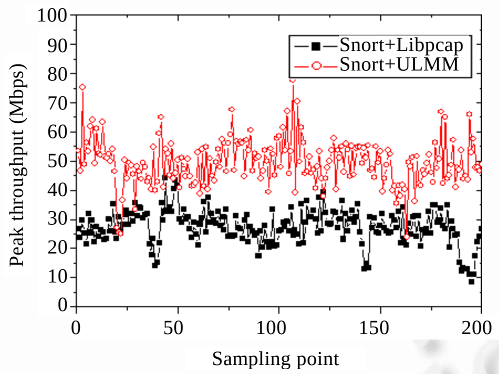

Fig.13 Processing performances of two intrusion detection systems in actual environment

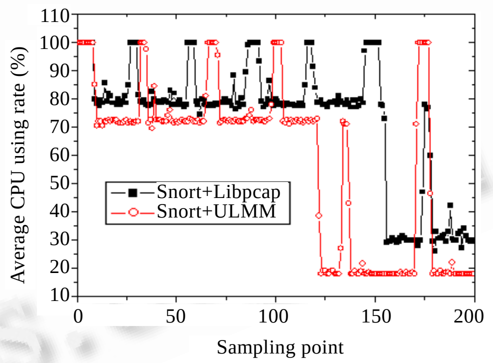

Fig.14 CPU using rates of two intrusion detection systems in actual environment

5 Conclusion

To meet the need of real intrusion analysis in high traffic network, this paper designs and implements a scalable efficient intrusion monitoring architecture (SEIMA) for intrusion detection. In SEIMA, the load balance algorithm based on connection round robin scheduling is employed to split network data; an efficient zero-copy user-level messaging mechanism (ULMM) is implemented by means of the statically allocated message buffer, static address translation table and special buffer management mechanism; a multi-rule user-level packet filter at the user layer (ULPF) is built by using an automatic algorithm. The application in an actual environment indicates that SEIMA is very practical for network security applications.

References:

[1] Meoch R. Snort-Lightweight intrusion detection for network. In: Valian P, Toddk K, eds. Proc. of the 13th System Administration Conf. Seattle: USENIX Association, 1999. 229−238.

[2] Kruegel C, Valeur F, Vigna G, Kemmerer R. Stateful intrusion detection for high-speed networks. In: Abadi M, Bellovin S, eds. Proc. of the 2002 IEEE Symp. on Security and Privacy. Berkeley: IEEE Computer Society, 2002. 266−274.

[3] Libpcap. 2002. http://www.tcpdump.org/release/libpcap-0.7.2.tar.gz

[4] Yang W, Fang BX, Yun XC, Zhang HL. Research and improvement on the packet capture mechanism in Linux for high-speed network. Journal of Harbin Institute of Technology (New Series), 2005,12(5):494−499.

[5] Welsh M, Basu A, von Eicken T. Incorporating memory management into user-level network interfaces. Technical Report, TR97-1620, Cornell University, 1997. http://www.cs.cornell.edu

[6] Eicken V, Vogels W. Evolution of the virtual interface architecture. IEEE Computer, 1998,31(11):61−68.

[7] Dubnicki C, Iftode L, Felten EW, Li K. Software support for virtual memory-mapped communication. In: Anderson T, Culler D, eds. Proc. of the 10th Int'l Parallel Processing Symp. (IPPS'96). Washington: IEEE Press, 1996. 372−381.

[8] McCanne S, Jacobson V. The BSD packet filter: A new architecture for user-level packet capture. In: Begel A, McCanne S, SLGraham, eds. Proc. of the 1993 Winter USENIX Technical Conf. San Diego, 1993. 259−269.

[9] Sekar RC, Ramesh R, Ramakrishnan IV. Adaptive pattern matching. SIAM Journal on Computing, 1995,24(6):1207−1234.

YANG Wu was born in 1974. He is an associate professor of Harbin Engineering University. His current research areas are network security, etc.

YUN Xiao-Chun was born in 1971. He is a professor of the Harbin Institute of Technology and a CCF member. His current research areas are computer network, etc.

FANG Bin-Xing was born in 1960. He is a professor of the Harbin Institute of Technology and a CCF senior. His research areas are computer network and information security, etc.