A Parallel Intrusion Detection System for High-Speed Networks

Haiguang Lai[1], Shengwen Cai[1], Hao Huang[1], Junyuan Xie[1], and Hui Li[2]

[1] Nanjing University, Nanjing, China,
lite@263.net,
[2] PLA University of Science and Technology

Abstract. The process speed of network-based intrusion detection systems (NIDSs) is still low compared with the speed of networks. As a result, few NIDS is applicable in a high-speed network. A parallel NIDS for high-speed networks is presented in this paper. By dividing the overall traffic into small slices, several sensors can analyze the traffic concurrently and significantly increase the process speed. For most attacks, our partition algorithm ensures that a single slice contains all the evidence necessary to detect a specific attack, making sensor-to-sensor interaction unnecessary. Meanwhile, by making use of the character of the network traffic, the algorithm can also dynamically balance all sensors' loads. To keep the system as simple as possible, a specific sensor is used to detect the scan and the DoS attack. Although only one sensor is used for this kind of attacks, we argue that our system can still provide high process ability. ...

1 Introduction

NIDSs detect attacks by sniffing and analyzing network packets [1,2]. Because of the dramatic increase of the network speed, there is an urgent need for higher process ability of NIDSs. Currently, most NIDSs can cope with the network traffic at a speed up to 100-200 Mbps [3]. There is no problem when they are used in a 100 Mbps shared network. However, It is easy to observe gigabit traffic on the mirror port of the switch in a 100 Mbps switched network. The NIDSs' weak process ability is mainly because that the analysis of network packets needs much computing. So it seems that the problem could be resolved if the processors' speed is increased. Unfortunately, the speed of networks increases faster than the speed of processors. It's impossible to keep up with the speed of networks by just increase the CPU's speed of NIDSs.
To resolve the problem, several sensors could be used to process the traffic concurrently. Since each sensor only processes one part of the traffic, the whole system can process the traffic at higher speed. The key of such system is how to split the network traffic into slices as equally as possible and not loss any evidence necessary for attack detection. A simple, effective parallel intrusion detection system for high-speed network is presented in the paper. The system's partition algorithm provides good splitting equality and makes sensor-to-sensor interaction completely unnecessary.

M. Jakobsson, M. Yung, J. Zhou (Eds.): ACNS 2004, LNCS 3089, pp. 439–451, 2004.
© Springer-Verlag Berlin Heidelberg 2004

2 Related Work

Christopher Kruegel et al design a high-speed NIDS, in which the traffic is divided into slices and each slice is processed by one or more sensors [4]. The partitioning is done so that a single slice contains all the evidence necessary to detect a specific attack. The architecture of the system is showed in Fig. 1.

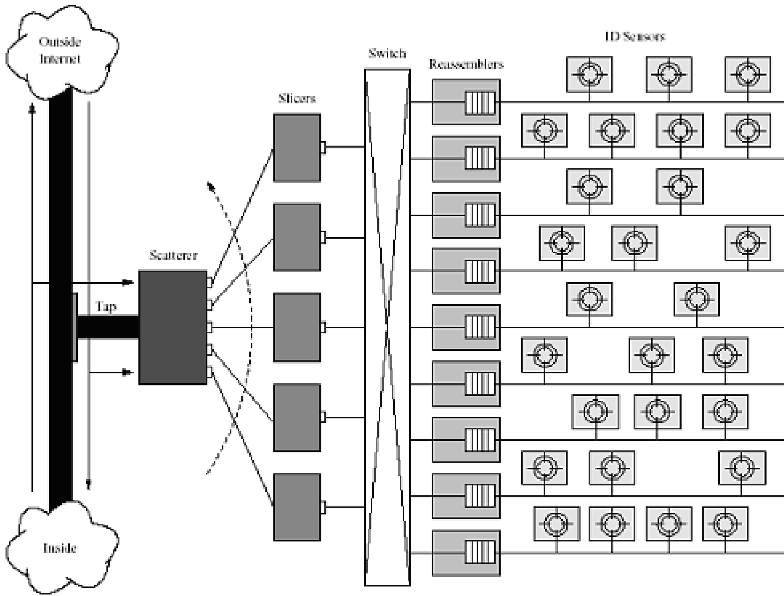

Fig. 1. Architecture of the high-speed intrusion detection

The scatterer only scatters frames in a round-robin fashion to guarantee high speed. The task of the slicers is to route the frames they receive to the sensors that may need them to detect an attack. The reassemblers are responsible to reassemble the possibly disordered frames. The system's throughput nearly reaches 200 Mbps in their experiment.

The paper mainly discusses how to divide the network traffic to avoid losing the evidence for intrusion detection. But the author does not present a feasible method to equally divide the traffic although he shows that the load balancing can be done by dynamically change the slices' filters. In the experiment, the traffic is statically divided according to the address range. Besides, the system is also complex, which needs many devices and has to reassemble the disordered frames. All of these limit its application in the real world.

The traffic splitter is the key of the parallel intrusion detection system. Charitakis et al examine a splitter's architecture in their paper [5]. Two methods are used to improve the system's performance. The first is the use of early filtering where

a portion of the packets is processed on the splitter instead of the sensors. The second is the use of locality buffering, where the splitter reorders packets in a way that improves memory access locality on the sensors. The experiment shows that these methods do improve the performance. However, since early filtering and locality buffering may cause some packets been dropped or reordered, it is impossible for the sensors to do state analysis which is important to improve the detection accuracy. In addition, the author splits the traffic by simply hashing on flow identifiers and does not discuss the problem of load balancing in the paper.

3 Parallel Intrusion Detection System

3.1 System Structure

The system structure determines the data flow and control flow in the system. A good structure is able to guarantee the efficiency of partitioning traffic and provide good scalability. The system's process ability can be increased by simply adding more sensors in the system. Besides, a simple structure is more applicable than a complex structure when they provide the same functions. We try to design such a good and simple structure, which is showed in Fig. 2.

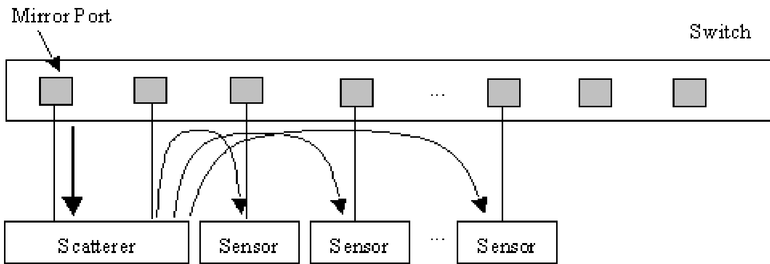

Fig. 2. Structure of the parallel intrusion detection

The scatterer is equipped with two network cards. One card is used to receive network packets from the switch's mirror port and the other one is used to send out the packets. Whenever a packet is received, the scatterer decides which sensor is responsible to process it according to the partition algorithm. Then the packet is sent out after changing its destination MAC address to the sensor's MAC address. The switch's mirror port mirrors all ports' traffic except the ports to which the scatterer and the sensors connect. The pairs of the sensors' MACs and the ports to which they connect are stored in the switch's "MAC-Port" table as static entries so that the broadcasting process of establishing the table is avoided.

This is a very simple structure since only a few devices are needed. To increase the system's process ability, all that we need to do is to add more sensors and

configure the switch. Our preliminary experiment has proved that the system is effective, scalable and efficient.

3.2 Partition Algorithm

The partition algorithm is the core of the system. An ideal partition algorithm should satisfy these requirements:

1. The algorithm divides the whole traffic into slices with equal sizes.
2. Each slice contains all the evidence necessary to detect a specific attack.
3. The algorithm is simple and efficient.

The first requirement is the key to guarantee the system's performance. If one slice is much larger than others are, the sensor that processes the slice will become the bottleneck of the system and waste other sensors' process abilities. The second requirement assures that each sensor can detect attacks without any interaction so that the system's complexity is greatly reduced. The third requirement is also for the performance. It is obvious that a simple and efficient algorithm is easy to archive a high throughput. Unfortunately, in practice, it is very hard to satisfy all of these requirements. For example, the round-robin partition algorithm, which sends the packets to the sensors in turn, completely satisfies requirement 1 and requirement 3. However, because the algorithm does not consider any character of the packets, it is very possible that several packets relevant to the same attack are sent to different sensors. As a result, no sensor has enough information to detect the attack. The round-robin algorithm does not satisfy requirement 2. On the other hand, to satisfy requirement 2, it is difficult to partition the traffic equally and keep the algorithm simple. After studying the network attacks, we design a simple algorithm that satisfies requirement 2 and guarantee that the slices have the nearly same sizes.

There are two kinds of network attacks. The attacks in the first category are those that can be detected after inspecting all the packets belonging to one TCP/UDP connection (Although there is no connection using UDP, we call it connection for convenience). Most attacks fall into this category, such as the attacks making use of the bugs of the programs. On the contrary, several connections have to be inspected to detect the attacks in the second category. Scan and DoS fall into this category.

For the first kind of attacks, the sensor would be able to detect the possible attacks without other information if all packets belonging to the same TCP/UDP connection are sent to it. So, no sensor-to-sensor interaction would be needed if partitioning according to TCP/UDP connections.

One TCP/UDP connection is identified by source IP address, source port, destination IP address, and destination port. Here, they are expressed respectively as *src_ip, src_port, dst_ip*, and *dst_port*. The TCP/UDP connection is defined as $conn(src_ip, src_port, dst_ip, dst_port)$. U denotes the aggregation of the packets belonging to the same partition. The constraint of the partition to satisfy requirement 2 is:

Constraint. *For any packets* p_i, p_j, $p_i \in conn$, $p_i \in U$, $p_j \in conn'$, $p_j \in U'$, *if* $conn = conn'$ *then* $U = U'$.

It is easy to prove that partitioning with any function of the source address, source port, destination address, and destination port is the sufficient condition of the constraint.
$f(src_ip, src_port, dst_ip, dst_port)$ denotes the function of the source address, source port, destination address, and destination port. Partitioning with the function means that:

Proof. For any packets p_i, p_j, $p_i \in conn$, $p_i \in U$, $p_j \in conn'$, $p_j \in U'$, if $f(p_i.src_ip$, $p_i.src_port$, $p_i.dst_ip$, $p_i.dst_port) = f(p_j.src_ip$, $p_j.src_port, pj.dst_ip, pj.dst_port)$ then $U = U'$.
Now, if $conn = conn'$ then $p_i.src_ip = p_j.src_ip$, $p_i.src_port = p_j.src_port$, $p_i.dst_ip = p_j.dst_ip$, $p_i.dst_port = p_j.dst_port$. So, $f(p_i.src_ip$, $p_i.src_port$, $p_i.dst_ip$, $p_i.dst_port) = f(p_j.src_ip$, $p_j.src_port$, $p_j.dst_ip$, $p_j.dst_port)$, then $U = U'$, the constraint is satisfied.

Since partitioning with the function of the source address, source port, destination address, and destination port guarantees detecting attacks without sensor-to-sensor interaction, a natural partition algorithm is to split the traffic with one or combination of these parameters. For instance, the address space of the protected network can be divided into several parts according to the destination address. One sensor is responsible for one part. Whenever the scatterer receives a packet, the scatterer checks the destination of the packet and forwards it to the sensor responsible for this address. However, although the interaction between sensors is avoided, it is hard to balance the loads of the sensors for the algorithm. Because the network traffic is dynamic, the static partition is unable to keep all parts equal. The dynamic adjustment of the partition is necessary to balance the loads of the sensors. This can be achieved by using an analyzer to collect and analyze the loads of the sensors. When finding one sensor's load is too high, the analyzer tells the scatterer to adjust the partition of the traffic so that the sensors' loads can be balanced. The resolution sounds good. However, to guarantee no evidence of the attacks is lost because of the load balancing, a sensor has to keep and exchange with other sensors the information of the connections currently processed. This is resource consuming and complex. Besides, the interaction of the analyzer and the sensors also increase the complexity of the system. In order to keep the system as simple as possible, no additional device is added in our design and the load balance is achieved by a special partition algorithm.
In the real networks, especially the large networks providing services, such as WWW and FTP, plenty of TCP/UDP connections are established and terminated all the time. Because the connections is established and terminated by the computers individually, the appearance of new connections is statistically random. As a result, there are always new connections during a short period. It is seldom that no new connection appears in a very long period. Therefore, the scatterer will frequently receive the packets belonging to the new connections

in a real network. For a new connection, all sensors are available to process its packets. The dynamic load balance is archived by choosing the sensor with the lightest load to process the new connection's packets. To measure the load of the sensor, the packets sent to every sensor are counted. The load is considered in direct proportion to the value of counting. The partition algorithm is described in details as following:

Algorithm 1. There are two tables on the scatterer. One is the table S that maps $src_ip \| src_port \| dst_ip \| dst_port$'s hash value k to the sensor's index ("$\|$" denotes concatenation). The other is the table C that maps the index of the sensor to the count of packets. C is used to record the number of the packets each sensor has processed. Initially, S is empty and the count is zero for every sensor in C. When the scatterer receives a packet, the packet is processed as following:

1. Calculate $src_ip \| src_port \| dst_ip \| dst_port$'s hash value k.
2. Search in S for the sensor to which k maps. If it is found, the packet is forwarded to that sensor and the sensor's count is increased by one in C. If it is not found, the packet is forwarded to the sensor with the lowest count. Meanwhile, an entry that maps k to the sensor is inserted to S and the sensor's count is creased by one in C.

In this algorithm, TCP/UDP connection is represented by the hash value k. The traffic is partitioned with k. As we have proved, partitioning with any function of the source address, source port, destination address, and destination port is the sufficient condition of the constraint. Therefore, partitioning with k satisfies the constraint. Using the hash value k instead of $src_ip \| src_port \| dst_ip \| dst_port$ is because that $src_ip \| src_port \| dst_ip \| dst_port$ needs 96 bits in total. If it were used as the index of the table S, there would be 296 entries in S. It is impossible in practice to implement a table with such huge size.
Algorithm 1 guarantees that the packets belonging to the same connection are processed by the same sensor so that the attacks in the first category will not be missed after partitioning. For the attacks in the second category, the packets relevant to one attack appear in different connections. Using algorithm 1, they may be sent to several sensors so that no sensor will have adequate evidence to detect the attack.
There is usually no significant signature for intrusion detection in a single packet belonging to the attack in the second category. Normal connection establishing or finishing packets can be used to probe whether some services are provided in the target host. Although some DoS attacks' packets do have certain signatures, it is usually not sufficient to make a conclusion that a specific attack occurs when a packet with the signature is detected. However, the attack's packets share some common characters. All packets of a scanning have the same source addresses while the destination addresses of the packets of a DoS attack are completely same. It is obvious that no scan and DoS attacks will be missed if all packets with the same source addresses are processed by the same sensor and all packets with the same destination addresses are processed by the same sensor.

Algorithm 1 is modified to ensure that the packets with the same source addresses are sent to the same sensor and the packets with the same destination addresses are sent to the same sensor. Obviously, no attack in the first category will be missed because the packets belonging to the same connection must have the same source or destination addresses. The following is the modified algorithm:

Algorithm 2. Three tables are used on the scatterer. S is the table used to map the source address to the index of the sensor. Table D is to map the destination address to the index of the sensor. Table C is used to record the number of the packets each sensor has processed. At first, S and D are empty and the counts in C are all zero. Whenever a packet is received, the following steps are executed:

1. Get the packet's source IP address *src_ip* and destination IP address *dst_ip*.
2. Search in S for the sensor to which *src_ip* maps.
3. If it (called A) is found, the packet is forwarded to A and the count of the sensor A is increased by one in C. Search in D for the sensor to which *dst_ip* maps. If it (called B) is found, the packet is forwarded to B and the count of B is increase by one in C. If B is not found, an entry that maps *dst_ip* to A is inserted to D.
4. If A is not found, Search in D for the sensor to which *dst_ip* maps. If it (called B) is found, the packet is forwarded to B and the count of B is increased by one. Meanwhile, an entry that maps *src_ip* to B is inserted to S. If B is not found, the packet is forwarded to the sensor with the lowest count. At the same time, an entry that maps *src_ip* to the sensor is inserted t to S and an entry that maps *dst_ip* to the sensor is inserted to D. The count of the sensor in C is increased by one too.

Algorithm 2 ensures that no evidence of the attacks will be missed after partitioning the traffic. Nevertheless, it brings new problems. First, the performance of the system is reduced because one packet may be sent to two sensors. Moreover, in the real networks, many connections may have the same destination address, a server's address for example. If algorithm 2 were used, all packets belonging to these connections would be sent to the same sensor and make its load too heavy. After some experiments, we find that algorithm 2 is completely unpractical.

Because of the attacks in the second category, it is difficult to keep the loads of the sensors balanced without the sensor-to-sensor interaction. To resolve the problem, we decide to modify the structure of the system. A single dedicated sensor is added to detect the attacks in the second category. Algorithm 1 is still used as the partition algorithm on the scatterer. The structure of the new system is showed in Fig. 3.

The sensors of type A are only responsible to detect the attacks in the first category. Sensor B is only used to detect the attacks in the second category. Like the scatterer, sensor B is connected to the mirror port of the switch. Therefore, it can receive all the packets of the network. No attacks in the second category will be missed as long as sensor B's speed is fast enough. In fact, it is not very difficult. For the attacks in the first category, the sensor has to reassemble the

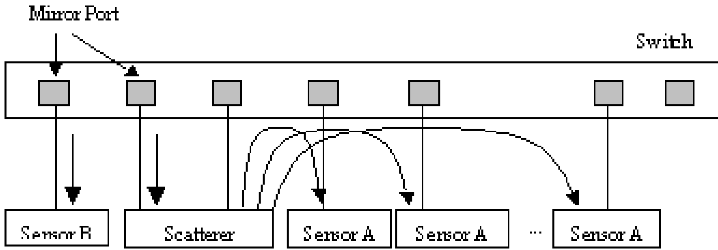

Fig. 3. Structure of the modified system

TCP streams and even the HTTP, FTP sessions. Then, the signatures of the attacks are searched in the payloads [6]. The whole process, especially the step of signature matching is time consuming. Therefore, the speed of intrusion detection is hard to keep up with the speed of the networks. However, to detect the attacks in the second category, the process is much simpler. Only the heads of the packets need to be checked. Since the flags of the packets' heads are limited, it is easy to detect the attacks at a high speed. R. Sekar designs a method to detect the attacks in the second category [7]. The method guarantees a high detection speed by translating the attacks' patterns into finite-state automata. Their system participated in an intrusion detection evaluation organized by MIT Lincoln Labs, where their system worked well at 500 Mbps. Therefore, the system structure showed in Fig. 3 is feasible. At least, the structure is still effective at 500 Mbps if Sekar's method is used in sensor B.

4 Experiment

In our prototype, the scatterer is a PC (Pentium4 1.8 G 512 M) equipped with two Intel8254GC Gigabit network cards, running Linux 2.4.18-13 (Redhat 7.3). The switch is Intel NetStructure4701T. The sensor for the detection of the attacks in the second category is not implemented in the prototype. The major purpose is to evaluate our partition algorithm.

To improve the performance of scattering, we implement the forwarding of the packets in the Linux kernel [8]. In more detail, the entries of the network protocols' process functions are stored in the array `packet_ptype_base`. First, we use function `dev_remove_pack()` to remove all protocols' functions for the array. Second, Our process function is inserted to `packet_ptype_base` by `dev_add_pack()` so that the packets will be processed by our function immediately after it is processed by the network card driver. According to the partition algorithm, the destination MAC addresses of the packets are changed and the packets are sent out by `dev_queue_xmit()`. Since the sensors process only IP packets, all packets except IP packets are dropped by the scatterer to further increase the scattering speed. The hash function used in the scatterer is the FNV hash, which is fast

while maintaining a low collision rate [9].The last 24 bits of the hash value k is used as the index of table S. As a result, the size of table S is 16 M bytes. The experiment consists of two parts. The effectiveness and the performance are evaluated respectively.

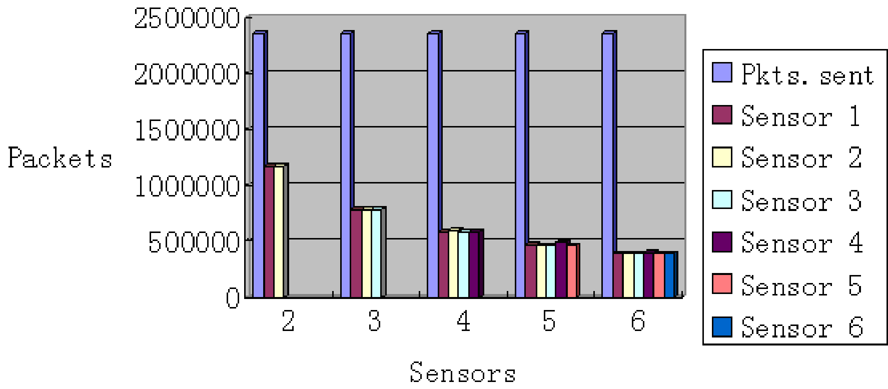

Fig. 4. Counts of scattered packets

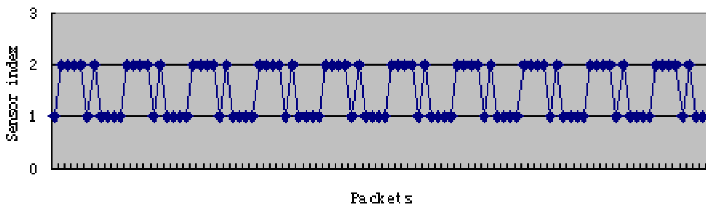

Fig. 5. Scatter packets to two sensors

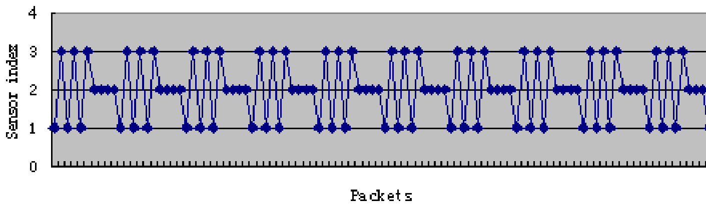

Fig. 6. Scatter packets to three sensors

4.1 Effectiveness

The evaluation of the effectiveness is to examine whether the traffic could be divided into slices with nearly the same size by the partition algorithm. We use tcpreplay to replay the traffic produced by MIT Lincoln Labs as part of the DARPA 1999 IDS evaluation [10]. The scatterer is configured to scatter packets to 2∼6 sensors. The number of the packets sent to the sensors is counted in the scatterer. Figure 4 and table 1 show the result.

Table 1. Counts of scattered packets

	2 sensors	3 sensors	4 sensors	5 sensors	6 sensors
Pkts. sent	2356503	2356503	2356503	2356503	2356503
Sensor 1	1169906	779103	581735	468647	388624
Sensor 2	1169900	779101	595170	459843	388649
Sensor 3		781553	581710	459841	388625
Sensor 4			581708	490155	393249
Sensor 5				461229	390935
Sensor 6					389588

As the result shows, the packets are almost equally scattered to all sensors. However, the result only demonstrates that the total numbers of the packets received by the sensors are nearly the same. It is also important that the instantaneous speed of the traffics scattered to the sensors are nearly equal, which means that the packets scattered to one sensor are nearly as many as those scattered to other sensors in a short period. Figure 5∼Fig. 9 show the process that the scatterer scatters packets. As the figures demonstrate, the packets are sent to different sensors in turn. It is not observed that all packets are sent to one sensor during a long period.

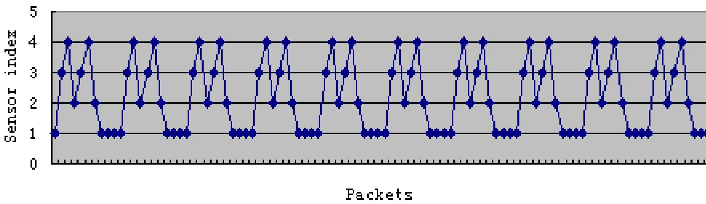

Fig. 7. Scatter packets to four sensors

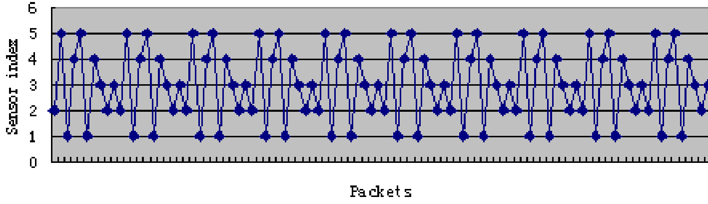

Fig. 8. Scatter packets to five sensors

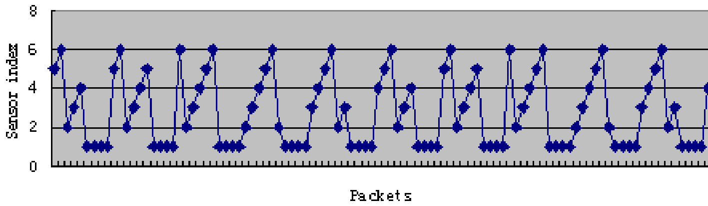

Fig. 9. Scatter packets to six sensors

4.2 Performance

Because the maximum amount of the traffic tcpreplay can generate is 160 Mbps, A SmartBits 600, which can generates faster traffic, is used to evaluate the performance. The percentage of the packets processed by the scatterer is recorded when the packets are sent out by the SmartBits with different size. The result is showed in Fig. 10 and table 2.

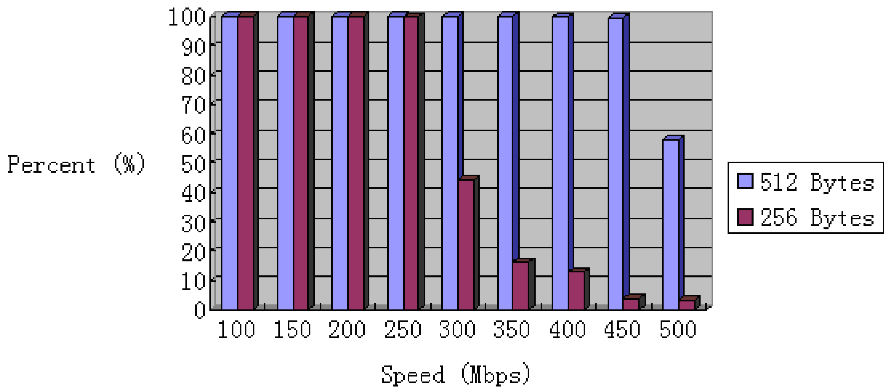

Fig. 10. Percentage of the packets processed

Table 2. Percentage of the packets processed

Speed (Mbps)	100	150	200	250	300	350	400	450	500
512 Bytes	100%	100%	100%	100%	100%	99.977%	99.848%	99.588%	58.132%
256 Bytes	100%	100%	99.968%	99.708%	44.341%	16.251%	12.969%	3.921%	3.533%

5 Conclusion and Future Work

A parallel intrusion detection system for high-speed networks is presented in the paper. The high-speed traffic is divided into several low-speed traffics, which are processed respectively by many sensors. Because of the parallel process, the system's performance is much better than the performance of one sensor. For all attacks except the scan and the DoS, the partition algorithm guarantees that the packets sent to each sensor contain all evidence necessary to detect a specific attack, making the complex sensor-to-sensor interaction unnecessary. Meanwhile, the algorithm provides the ability of dynamically balance the loads of the sensors. In more detail, the algorithm always chooses the idlest sensor to process the packets belonging to a new connection. In addition, the algorithm is very simple and efficient. For the scan and DoS attacks, it is difficult to keep the sensors' loads balanced while making sensor-to-sensor interaction unnecessary. Therefore, we use a specialized sensor to detect them and explain that the system's performance can still keep high despite only one dedicated sensor is used. Our preliminary experiment proves that the parallel intrusion detection system is of high performance and scalability. Our approach also has some shortages. The scatterer is the bottleneck of the system and the partition algorithm consumes much memory. Future work will include implementing the partition algorithm in hardware and the research on parallel detecting the scan and DoS attacks.

References

1. Rebecca Bace, Peter Mell: Intrusion Detection Systems. NIST Special Publication on Intrusion Detection Systems. (2001)
2. G. Vigna, R. Kemmerer: NetSTAT: A Network based Intrusion Detection Approach. Computer Security Applications Conference. (1998)
3. Bob Walder: Gigabit IDS. http://www.westcoast.com/artframe_report.html. (2003)
4. Christopher Kruegel, Fredrik Valeur, Giovanni Vigna, Richard Kemmerer: Stateful Intrusion Detection for High-Speed Networks. IEEE Symposium on Security and Privacy. (2002)
5. I. Charitakis, K. Anagnostakis, E. Markatos: An Active Traffic Splitter Architecture for Intrusion Detection. 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems. (2003)
6. V. Paxson: Bro: A System for Detecting Network Intruders in Real-Time. The 7th USENIX Security Symposium. (1998)

7. Sekar, R., Guang, Y., Verma, S., Shanbag, T.: A High-Performance Network Intrusion Detection System. ACM Symposium on Computer and Communication Security. (1999)
8. kossak: Building Into The Linux Network Layer. Phrack Magazine. **9**(1999)
9. Glenn Fowler, Phong Vo., Landon Curt Noll: Fowler / Noll / Vo (FNV) Hash. http://www.isthe.com/chongo/tech/comp/fnv/.
10. MIT Lincoln Laboratory: DARPA Intrusion Detection Evaluation. http://www.ll.mit.edu/IST/ideval/. (1999)