Copyright © 2018 Tech Science Press CMC, vol.56, no.1, pp.61-72, 2018

A Distributed Intrusion Detection Model via Nondestructive Partitioning and Balanced Allocation for Big Data

Xiaonian Wu[1, *], Chuyun Zhang[3], Runlian Zhang[2], Yujue Wang[2] and Jinhua Cui[4]

Abstract: There are two key issues in distributed intrusion detection system, that is, maintaining load balance of system and protecting data integrity. To address these issues, this paper proposes a new distributed intrusion detection model for big data based on nondestructive partitioning and balanced allocation. A data allocation strategy based on capacity and workload is introduced to achieve local load balance, and a dynamic load adjustment strategy is adopted to maintain global load balance of cluster. Moreover, data integrity is protected by using session reassemble and session partitioning. The simulation results show that the new model enjoys favorable advantages such as good load balance, higher detection rate and detection efficiency.

Keywords: Distributed intrusion detection, data allocation, load balancing, data integrity, big data.

1 Introduction

Intrusion detection system (IDS) is an important tool in protecting network security [Denning (1987)]. With the fast development of network technology, big data age arrived [Gupta and George (2016)], which brings many challenges to traditional IDS. Due to the huge volume of data and potential attacks in big data, it is rather difficult to provide effective security support for traditional IDS. In fact, traditional IDS has many disadvantages such as delaying response, reducing detection rate, thus it cannot effectively complete the detection work in big data environment. To solve these issues, distributed instruction detection system (DIDS) was introduced [Konstantinos, Ioannis and Spiros (2006); Genge, Haller and Kiss (2016)]. Although many DIDS schemes have proposed with different approaches, they have many shortcomings and need to be improved. For instance, data integrity may be destroyed when data is inappropriately partitioned, and the imbalance of data allocation may affect performance. Therefore, it is an urgent task to improve the efficiency and detection rate of IDS.

[1] Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China.

[2] Guangxi Key Laboratory of Cryptography and Information Security, Guilin University of Electronic Technology, Guilin 541004, China.

[3] Guangxi Wireless Broadband Communication and Signal Processing Key Laboratory, Guilin University of Electronic Technology, Guilin 541004, China.

[4] School of Information Systems, Singapore Management University, Singapore 178902, Singapore.

* Corresponding Author: Xiaonian Wu. Email: xnwu@guet.edu.cn.

The challenge work in current IDS is to improve data processing speed so that it can successfully match the speed of network transmission and processing for massive data. In this case, the key point is to look for some appropriate data partition algorithms to partition data, and efficiently process data. Kruegel et al. [Kruegel, Valeur, Vigna et al. (2002)] used a slicing mechanism to process data in a parallel mode. Later, Charitakis et al. [Charitakis, Anagnostakis and Markatos (2003)] introduced an active traffic splitter based on hash algorithm. In 2005, an algorithm is presented by the idea of priority to partition and distribute the data to the lightest load node [Jiang, Song and Dai (2005)]. Also, a hash mechanism is used to distribute network traffic load [Schaelicke, Wheeler and Freeland (2005)]. Recently, a distributed outlier detection algorithm for computing outliers is proposed [Wang, Shen and Mei (2016)]. Notice that data integrity in above methods could be destroyed when the traffic is inappropriately divided into subsets. Moreover, to ensure data integrity, a partition algorithm [Lai, Cai and Huang (2004)] is introduced to divide traffic into small slices by address, port and attack types. Furthermore, Liu et al. [Liu, Xin, Gang et al. (2008)] investigated the issue using the independent neural network. Recently, Cheng introduced an abnormal network flow feature sequence prediction approach to detect DDoS attacks in big data environment. However, the problem is still not well solved.

On the other hand, data partitioning has been widely used in parallel computing and data processing in various fields, to raise throughput and provide load balancing [Soklic (2002); Rosas, Sikora and Jorba (2014); Wang, Chen, Li et al. (2016)]. Although the throughput of system is increased by data partition and load policy, there still exists a bottleneck in IDS system.

In this paper, a new distributed intrusion detection model is proposed based on nondestructive partitioning and balanced allocation for big data (DIDS-NPBA). Specifically, a data allocation approach is introduced based on capacity and load to achieve local load balance, a dynamic load adjustment strategy is adopted to maintain global load balance of cluster and data integrity is protected by reassembling TCP session and session allocation policy.

The rest of this paper is organized as follows. A brief introduction and overview of DIDS-NPBA model are given in Section 2. The process about session assembly and data classification are described in Session 3. Section 4 presents a data allocation approach and a nondestructive data partitioning algorithm. The experimental results of the proposed model are shown in Section 5. Finally, the conclusions and future work are discussed in Section 6.

2 DIDS-NPBA system model

In high-speed network, massive amount of data makes traditional IDS difficult to detect attacks. Hadoop is an open-source framework and is used for large scale data processing. In this paper, our DIDS-NPBA is built on Hadoop to detect big data. As shown in Fig. 1, DIDS-NPBA consists of data acquisition, control center, data storage and data detection, where control center contains system monitoring, task scheduler and alarm response. According to Hadoop framework, control center acts as the master node, whereas other components play as slave nodes and communicate with MapReduce, which will be described in Section 5.1.

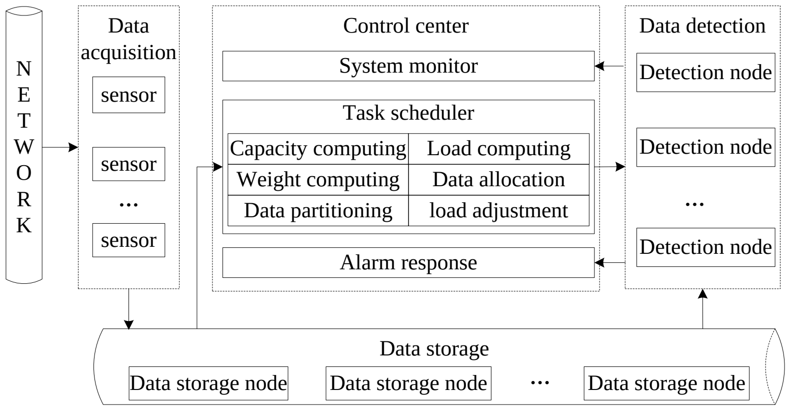

Figure 1: The DIDS-NPBA model

The mechanism of DIDS-NPBA is as follows. In Fig. 1, data acquisition is to collect network packets using many independent sensors. Packets are reassembled and classified in sensors. The data storage nodes receive and store data from sensors and submit summarizing information to control center.

System monitoring module monitors the state of detection nodes, including utilization ratio of node CPU/RAM/hard disk, and network bandwidth. Then the information is sent to task scheduler. According to the information and a data allocation strategy based on capacity and load (DAS-CL), task scheduler partitions and allocates data to detection nodes. Alarm response module receives and displays alarms coming from detection nodes.

Data detection module is used to detect and analyze data, which consists of many distributed data detection nodes in parallel. Each detection node reads the allocated data files from the corresponding data storage node, analyses and detects data, and sends alarms to control center.

3 The work of data assembly and classification detection

The TCP session assembly is to assemble and restore session from the interrelated network packets. A TCP link is determined uniquely by a tuple <source IP, source port, destination IP, destination port>. The hash value calculated from such tuple can be used as a unique identification of TCP link. Combining the hash value with linking number in packet, each TCP session can be restored from the captured packets. Then in sensors, each TCP session is stored into a data file named after its hash value. To protect data integrity, each session file would not be permitted to be split out when files is partitioned.

In IDS, more feature rules would be helpful to raise the detection rate. However, the growth of feature rules increases detection time. Thus, we introduce a classification detection method to reduce detection time by cutting down feature rules of IDS, without affecting the detection rate of IDS.

First, the session files are classified according to application protocols. For example, assuming there are *k* class protocols, the files would be divided into *k* data groups. Then, all detection nodes are divided into *k* node groups in the same way, and the feature rules corresponding to the protocol are retained in each node group and the other rules are cut down. Finally, each data group is assigned to a node group with the same protocol. In this way, the feature rules can be reduced, which hardly affect the detection rate of IDS since these feature rules meet the pending data.

4 Data allocation strategy and data partition algorithm

4.1 Capacity evaluation for detection nodes

The capacity of nodes is determined by many performance indexes, which can be collected by the system monitoring module. Assume the evaluation indexes for node capacity mainly include node CPU, memory and network bandwidth, since the work focuses on computation and data transmission. We only evaluate the detection nodes which are used to detect data. Let $CAP(i)$ represent the capacity of node *i*, which is defined as follows:

$$CAP(i) = a_1 \times CPU_i + a_2 \times M_i + a_3 \times N_i \tag{1}$$

where CPU_i is the product of CPU number and CPU frequency of node *i*, M_i is the memory capacity, N_i is network bandwidth, and a_1, a_2, a_3 are the weighting factors on node capacity with $a_1+a_2+a_3=1$.

To evaluate the diversity of memory among heterogeneous nodes, M_i is computed by memory size, frequency and CAS latency (CL) of memory, where these parameters are normalized by using Min-max linear function conversion method. The computation of M_i avoids the influence from the discrete value of different indexes. Similarly, CPU_i, N_i in formula (1) are normalized with the same linear method.

The heterogeneity of detection node capacity makes it difficult to control the data granularity allocated to each node when big data is split. If all nodes are assigned datasets with the same size, the nodes with weak capacity would be bottleneck. Here, we introduce a concept about node capacity factor (*CF*) to measure the capacity relationship of nodes as follows.

Definition 1. *CF* is mainly used to measure or weigh capacity ratio of a node in cluster. Assume there are *n* nodes in a cluster, $CAP(i)$ is the capacity of node *i*, and node *j* ($1 \le j \le n$) has the lowest capacity $CAP(j)$ in all nodes. *The CF* of node *i* can be computed as follows:

$$CF(i) = \frac{CAP(i)}{Min(CAP(1), CAP(2), ..., CAP(j), .., CAP(n))} \tag{2}$$

The node with the lowest capacity is used as a benchmark to measure or weigh other node in cluster. Moreover, *CF* reflects the ratio among nodes in cluster, which can be helped to calculate the weight for data allocation.

4.2 Load evaluation for detection nodes

The load of nodes is another important factor affecting the data processing efficiency. The heavier load of a node, the more time is needed for data detection. In order to evaluate node load, the utilization of CPU, memory and network bandwidth of nodes are collected by the system monitoring module in real time. Suppose $L(i)$ is the load of node i at a certain moment, we define:

$$L(i) = b_1 \times U(CPU_i) + b_2 \times U(M_i) + b_3 \times U(N_i) \tag{3}$$

where $U(CPU_i)$, $U(M_i)$, $U(N_i)$ are the CPU, memory and network bandwidth utilization of node i, and b_1, b_2, b_3 are the weight factors satisfying $b_1+b_2+b_3=1$.

If the load of some node is changed, then the data granularity allocated to this node should be changed accordingly. For instance, the overload nodes will no longer be assigned data. Here, we give another definition about the load factor (*LF*) to estimate the load degree of heterogeneous nodes.

Definition 2. *LF* is mainly used to measure the degree of node load. Suppose the light-load threshold of a node is α and its overload threshold is β. *The LF* of node i can be computed as follows:

$$LF(i) = \begin{cases} 1 + \dfrac{|L(i) - \alpha|}{L(i)}, & L(i) \le \alpha \\ 0, & L(i) \ge \beta \\ \dfrac{|L(i) - \beta|}{L(i)}, & \alpha < L(i) < \beta \end{cases} \tag{4}$$

where *LF* is a piecewise function such that its value changes along with the node load. Thus, *LF* is the other key factor to evaluate data allocation.

4.3 Data allocation strategy in a node group

In the proposed DAS-CL, the contribution of each node is quantified in a node group. Assuming there are *m* nodes in node group *k*, and the total GDB_k data are submitted to node group *k*. *The* weight factor $R(i)$ of node *i* in group *k* can be computed as follows:

$$R(i) = \frac{CF(i) \times LF(i)}{\sum_{i=1}^{m} (CF(i) \times LF(i))} \tag{5}$$

From Eq. (5), the stronger capacity and the lighter load of node *i*, it has the greater weight or contribution in node group *k*.

Suppose $D(i)$ is the assigned data size to node *i* from GDB_k, which can be computed as follows:

$$D(i) = GDB_k \times R(i) \tag{6}$$

According to formula (6), each node will be allocated a data set conforming to its capacity and load. Thus, the resource utilization is raised and the bottleneck can be avoided, which means that a better local load balance can be achieved in node group.

4.4 Data partition

In the process of data partition, files allocated to node *i* may be roughly $D(i)$, but does not equal to $D(i)$, since the size of each file is uncertain. If a file is split out to meet $D(i)$, the data integrity would be undermined. Thus, each file would not be permitted to be split out, as discussed in Section 3.

An approximate allocation way is used to partition data. A fluctuation range closed to $D(i)$ is set to 5 MB, it has little influence on node load when compared with the massive data. Then a file or many files from GDB_k with size closed to $D(i)$ will be allocated to node i, and the file information including the address of data storage node (i.e. the real location of storing data file), file directory and file name, etc. will be sent to node i by the task scheduler. Node i will read the assigned data files from the data storage node, detect the data, and send alarms to the alarm node.

4.5 Dynamic adjustment for load balancing among node groups

The data produced by the Internet applications is varying with time. For instance, during some period, an application may be used frequently, thus the produced data would be increased, whereas other data may be reduced. This changing could be able to affect the load of node groups, even cause load imbalance among node groups. Therefore, we present a dynamic load adjustment method among node groups to maintain global balance in the cluster.

We need to evaluate the load of different node groups. In each node group, each node may have different capacity and load. We introduce a new concept regarding the load of node group (GL) to smooth the differences.

Definition 3. GL is used to measure the average load of all nodes in a node group, which can be computed by the load and capacity of nodes. Suppose there are m nodes in node group k, the load $GL(k)$ of node group k is defined as follows:

$$GL(k)=\sum_{i=1}^{m}\frac{CF(i)\times L(i)}{\sum_{i=1}^{m}CF(i)} \tag{7}$$

The load dynamic adjustment method among node groups is as follows:

Step 1. Compute GL of each node group. The light-load threshold of node group is α' and its overload threshold is β'. All the overload node groups are lined in an overload queue, and all the light-load node groups are lined in a light-load queue.

Step 2. The task scheduler counts up the classified data coming from data storage nodes, and assigns each dataset to a node group by protocols.

Step 3. Set a light dataset threshold δ, which is considered to have less impact on the load of node group even if the node group is overload. Assuming a node group is assigned data of size GDB, if $GDB \leq \delta$, then it will be removed from the overload queue.

Step 4. Suppose two queues are all not empty. From the first node group A in the light-load queue, node i is searched, who has completed detection tasks and has lighter load, and its $CAP(i)$ is not the largest in the group. Node i is migrated to the first node group B in the overload queue, and the feature rules of IDS in the migrated node i are changed conforming to the new node group. Then, node groups A and B are removed from two queues, respectively.

Step 5. If the light-load queue is empty and the overload queue is not empty, then the assigned data of all node groups in the overload queue are split out such that half data will be processed in the current round and the remaining will be processed in the next round. Then all node groups are removed from the overload queue.

5 Experimental result analysis

5.1 Experiment environment

Based on Hadoop, a prototype of DIDS-NPBA model is implemented with Java. Nodes running above function are deployed in campus network. For instance, sensors are distributed in different subnet and near switches. The system monitor unit uses open source tool Sigar to collect run-time information of each detection node. DAS-CL strategy and data partitioning are implemented and embedded in task scheduler in control center. The alarm response unit is used to count alarm results. In order to ensure the consistency and validity of data detection, all detection nodes equip with Snort system after detection nodes are grouped, feature rules in each detection node are configured to conform to protocols of its node group. In Hadoop cluster, control center plays as a Master node, which communicates with the slave nodes including data storage nodes and detection nodes using MapReduce.

In this section, we test the DIDS-NPBA model in the above environment in terms of load balancing, detection time and detection rate, and compare with two data allocation methods, that is, consistency hash algorithm [Karger, Lehman, Leighton et al. (1997)] and Rosas algorithm [Rosas, Sikora and Jorba (2014)].

The parameters in DIDS-NPBA model are set as follows:

(1) In formula (1) and (3), the computing power plays a major role in data detection, thus a_1, b_1 are set to 0.6, a_2, b_2 are set to 0.3, and a_3, b_3 are 0.1.

(2) In formula (4), according to the empirical data, the rated output of nodes is about 0.7~0.8, and referring to the experimental data of node load, α and β *are* set to 0.23 and 0.77, respectively.

(3) In Section 4.4, α' and β' are set to 0.3 and 0.7, respectively, which are different from α and β, since the average load of a node group can be impacted by more factors. And δ is set to 200 MB.

(4) The detection nodes count up to 20 heterogeneous nodes, whose capacities are computed as in Tab. 1.

5.2 Data preprocessing and grouping for detection nodes

We use DARPA2000 data set to conduct testing in this paper. Since the DARPA2000 data set is small, the data in inside, outside and DMZ environments from different dates are extended to 3 GB to simulate data acquisition in high-speed network. The extended 3 GB data set is reassembled and processed in sensors, and each session is stored in a file named after its hash value.

Through analyzing the above 3 GB data set, the total number of attacks is 15659. Moreover, the data components are as follows: The http protocol data is 1782 MB, the ftp data is 233 MB, and the telnet data is 522 MB. The rest 463 MB data are classified as mixed data. Thus, the total data set is divided into four categories, that is, HTTP, FTP, TELNET, and mixed data.

Accordingly, detection nodes are also divided into four node groups. The feature rules of Snort in each node are configured according to its group, except the mixed node group is with all original feature rules of Snort.

Initially, detection nodes are grouped as follows. HTTP node group is assigned 10 nodes (S1~S10), FTP node group has 2 nodes (S11~S12), TELNET node group has 4 nodes (S13~S16), and the mixed node group contains the resting 4 nodes (S17~S20). The capacity of nodes is shown in Tab. 1.

5.3 Validity test and analysis for the DIDS-NPBA model

In this section, the load balancing, detection time and detection rate of DIDS-NPBA model are investigated. The results are compared with that of consistency hash [Karger, Lehman, Leighton et al. (1997)] and Rosas et al. [Rosas, Sikora and Jorba (2014)] algorithm, which are listed in Tab. 1.

Table 1: The results of data allocation and detection with different methods

Node	*CAP*	*L*	*CF*	*Data size (MB)*			*Detection time (second)*			*Alarm number*		
				NPBA	Hash	Rosas	NPBA	Hash	Rosas	NPBA	Hash	Rosas
S1	2350	0.35	1.65	225	180	230	9.6	9.7	9.6	357	263	361
S2	2278	0.33	1.60	218	178	203	9.5	9.5	10.3	330	252	345
S3	2021	0.40	1.42	193	173	207	9.8	9.3	10.1	320	244	332
S4	2010	0.35	1.41	192	165	206	10	9.7	9.8	305	235	319
S5	1920	0.38	1.35	184	177	171	10.2	10.1	9.4	285	257	274
S6	1750	0.41	1.23	167	180	170	11.2	10.9	9.3	260	261	268
S7	1675	0.37	1.17	160	182	153	10.2	10.5	10.2	253	258	243
S8	1630	0.45	1.14	156	187	151	11	11.1	10.5	250	267	231
S9	1580	0.35	1.11	151	185	161	10.8	10.8	10.3	240	259	232
S10	1425	0.40	1	136	175	130	11.5	12.3	10.9	215	248	186
S11	1690	0.37	1.44	138	123	131	9.6	9.2	8.5	184	168	180
S12	1171	0.48	1	95	110	102	8.9	12.2	11.6	131	143	135
S13	2149	0.52	1.77	166	120	159	9	8.3	10.4	193	163	187
S14	1850	0.51	1.52	143	135	149	10	9.6	8.6	178	170	181
S15	1542	0.44	1.27	119	142	113	9.4	11	10.8	140	153	136
S16	1214	0.55	1	94	125	101	9.1	12.3	10.6	115	139	121
S17	2098	0.35	1.51	147	113	144	9.9	8.4	10.2	3698	3416	3691
S18	1786	0.40	1.28	120	126	129	9.4	9.6	10.5	3152	3024	3161
S19	1470	0.51	1.06	101	118	99	9.6	9.8	9.7	2590	1587	2602
S20	1390	0.58	1	95	106	91	9.7	11.3	10.3	2460	2508	2439
						total	197.8	205.6	201.6	15656	14015	15624

In Tab. 1, nodes are listed by grouping in turn, and each group has different background color. *CAP* represents node capacity, *L* is node load, *CF* is the node capacity factor, *Data size* is the partitioned and assigned data to each node, *Detection time* is the time spent by each node to detect the assigned data, and *Alarm number* is the attack number detected by each node.

(1) *Result analysis for load balancing experiments*

In parallel computing, the data assigned to each node in cluster should be proportionate to node performance [Soklic (2002)] to maximize resource utilization.

In Tab. 1, it can be seen that the difference of capacity (*CAP*) among nodes is great, whereas the load (*L*) among nodes are close. This can be evaluated by the standard deviation (*SD*). *SD* of *CAP* and *L* of 20 nodes are 335 and 0.076, respectively. In Tab. 1, *SD* of the data assigned to each node group by DIDS-NPBA, hash and Rosas, are computed. For HTTP, FTP, TELNET and mixed node group, there are 29.2, 30.4, 31, 23.4 in DIDS-NPBA, 6.3, 9.2, 9.9, 8.4 in hash, and 31.6, 20.5, 27.9, 24.9 in Rosas.

The above results show that, the capacity difference of nodes is great, whereas the difference of the data assigned to each node is small in consistency hash, which means that a node with weak capacity has to deal with the same data set like a strong node. In fact, the consistency hash partitions data without considering node capacity and load. Rosas algorithm adjusts the allocated data amount dynamically by the computing time that the node dealt with the allocated data, which matches the performance of nodes. For DIDS-NPBA, it can be seen that the data assigned to each node is proportionate to its capacity and load, which improves resource utilization and throughput of system, and the load balance of cluster system can also be maintained well.

(2) *Analysis of experimental results for detection time*

In Tab. 1, *SD* of detection time for DIDS-NPBA, Hash and Rosas, are 0.72, 1.19, 0.75, respectively. The results show that the difference of detection time spent by each node in DIDS-NPBA is smallest and its parallelism is the best. Consistency hash requires the longest detection time (12.3 s) for a single detection node due to imbalance data allocation, thus the parallelism is worst. Due to the balanced data allocation, the parallelism of Rosas is roughly the same to DIDS-NPBA. Note that the sum of detection time spent in DIDS-NPBA is the least, which shows that the system performance is improved.

(3) *Analysis of experimental results for detection rate*

DR is defined as the ratio between number *D* of the detected intrusion behaviors and number *T* of all intrusion behaviors, that is, $DR=D/T$.

It is shown in Tab. 1, *DR* of the hash is the lowest, i.e. 89%, which mainly lies in that the data integrity is damaged heavily when data is partitioned by mapping relationship between packets and virtual nodes. *DR* of the Rosas and DIDS-NPBA are near to 99%, because the data integrity is protected by TCP session assembly and balanced data allocation. Moreover, the original data that have not to be assembled based on TCP session, *DR* of the Rosas and DIDS-NPBA are near to 93%.

5.4 Experimental test for the dynamic load balancing adjustment among node groups

To evaluate the efficiency of our proposed algorithms, we construct experiments on five datasets, where each date set is about 3 GB as in Section 5.2. The components of five data sets are as follows: The first data set is same as that used in Section 5.2, and other four data sets are http data of 1382 MB, ftp data of 633 MB, telnet data of 522 MB, and the mixed data of 522 MB, respectively, with the total alarm 15590.

In the experiments, the 3 GB data set is allocated and detected by 20 nodes in parallel based on the DIDS-NPBA model, and the average detection time is about 10 seconds. Thus, the processing of a data set including partition, allocation and detect data is 10 seconds. Note that 20 detection nodes are still used, and the initial group and experimental results for the first data set are shown in Tab. 1. The experimental results for the other four data sets are summarized in Tab. 2, where the titles, L, Data, Time and Alarm, have same meaning as in Tab. 1.

Table 2: The experimental results of the dynamic load balancing adjustment strategy

Nodes	The second data set				Nodes	The third data set				The fourth data set				The fifth data set				L
	L	Data	Time	Alarm		L	Data	Time	Alarm	L	Data	Time	Alarm	L	Data	Time	Alarm	
S1	0.26	174	6.6	276	S1	0.30	199	8.6	316	0.28	199	9.1	316	0.30	199	9.3	316	0.30
S2	0.28	169	6.8	266	S3	0.29	171	8.7	272	0.29	171	9.3	272	0.29	171	8.6	272	0.29
S3	0.31	150	7.1	238	S4	0.24	170	9.5	270	0.31	170	9.7	270	0.33	170	9.7	270	0.33
S4	0.57	149	7.4	236	S5	0.25	163	10.7	252	0.62	163	10.2	252	0.65	163	11.1	252	0.62
S5	0.63	143	8	227	S6	0.33	148	11.9	239	0.61	148	11.3	239	0.63	148	10.9	239	0.63
S6	0.60	130	6.8	205	S7	0.31	141	10.8	224	0.60	141	11	224	0.61	141	11.3	224	0.64
S7	0.58	124	7.6	197	S8	0.35	137	11.6	215	0.57	137	10.8	215	0.65	137	11	215	0.65
S8	0.61	120	7.1	191	S9	0.31	134	11.3	212	0.60	134	11	212	0.61	134	10.9	212	0.67
S9	0.59	117	6.9	186	S10	0.34	120	10.8	191	0.63	120	10.9	191	0.64	120	11.3	191	0.64
S10	0.65	106	8.1	169	S2	0.35	345	13.9	415	0.70	345	13.7	417	0.66	262	8.9	353	0.68
S11	0.32	374	26.8	427	S11	0.73	288	10.8	368	0.69	288	10.7	368	0.67	219	10.8	298	0.62
S12	0.30	259	28.6	289	S12	0.87	0	0	0	0.89	0	0	0	0.42	152	10.5	213	0.65
S13	0.25	166	9.3	193	S13	0.29	166	8.9	193	0.27	166	9	193	0.33	166	9.5	193	0.30
S14	0.57	143	10.2	178	S14	0.60	143	10	178	0.59	143	10.4	178	0.57	143	10.2	178	0.62
S15	0.63	119	11	140	S15	0.59	119	10.9	140	0.64	119	11	140	0.59	119	10	140	0.58
S16	0.30	93	8.5	115	S16	0.27	93	8.3	115	0.31	93	8.5	115	0.30	93	9.1	115	0.34
S17	0.28	162	9.3	3698	S17	0.26	162	9.8	3698	0.31	162	9.3	3698	0.34	162	9.7	3698	0.22
S18	0.61	138	10	3152	S18	0.64	138	10.7	3152	0.59	138	10	3152	0.60	138	11.2	3152	0.63
S19	0.63	114	10.3	2590	S19	0.67	114	11.1	2590	0.65	114	11	2590	0.58	114	10.3	2590	0.57
S20	0.62	108	10	2460	S20	0.66	108	10.5	2460	0.63	108	10.7	2460	0.64	108	10	2460	0.67
		total	206.4	0				198.8	10.78			197.6	10.41			204.3	10.65	

Compared the second data set with the first one, it is easy to see that http data are decreased, whereas ftp data are increased. After the second data set has been split and allocated, the nodes in ftp node group were assigned with more data than the first time. As shown in Tab. 2, node S12 is overloaded at 0.87, and ftp node group is also overloaded at 0.79. However, the load of nodes in http node group is decreased, and the http node group has light load at 0.26. According to the dynamic load adjustment strategy, node S2 in http node group is migrated to ftp node group. Then, in ftp node group, node

S2 and S11 are assigned data for the third and fourth data sets, but node S12 is ignored since it has been overloaded. When the fifth data set is processed, node S12 is monitored. It is assigned data according to its capacity and load, in this way to return to normal. And the load of all nodes is also monitored after the fifth data set has been processed as shown at the most right column in Tab. 2.

With network data sharply changes, some problems would inevitably occur for IDS, such as the imbalance of system load, and the system performance and detection rate are degraded. The global load imbalance due to the sharp change of data component can be overcome in time following the dynamic load adjustment strategy, and the data processing efficiency, load balance and detection rate of the system are relatively stable during the process of continuous operation.

6 Conclusion

In view of the issue that data processing speed of existing IDS cannot match the speed of network transmission and processing for massive data, a novel distributed intrusion detection model DIDS-NPBA is proposed. More precisely, feature rules are decreased by classification detection method. The data integrity can be protected by session reassembly and nondestructive partitioning based on session files. Furthermore, the load balance of system is achieved by the dynamic load adjustment strategy and DAS-CL strategy. The simulation results demonstrate that our model has favorable advantages in providing better load balance, reducing detection time, and raising detection rate and throughput.

Acknowledgement: This article is supported in part by Guangxi Key Laboratory of Trusted Software of China (No. kx201622), Guangxi Key Laboratory of Cryptography and Information Security of China (No. GCIS201623, GCIS201705), Guangxi Key Lab of Wireless Wideband Communication and Signal Processing of China (No. GXKL061510, GXKL0614110), Guangxi Colleges and Universities Key Laboratory of cloud computing and complex systems of China (No. YF16205), and Innovation Project of Guangxi Graduate Education (No. YCSW2018138, 2017YJCX26).

References

Charitakis, I.; Anagnostakis, K.; Markatos, E. (2003): An active traffic splitter architecture for intrusion detection. *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems*, pp. 238-241.

Denning, D. E. (1987): An intrusion-detection model. *IEEE Transactions on Software Engineering*, vol. 13, no. 2, pp. 222-223.

Genge, B.; Haller, P.; Kiss, I. (2016): A framework for designing resilient distributed intrusion detection systems for critical infrastructures. *International Journal of Critical Infrastructure Protection*, vol. 15, pp. 3-11.

Gupta, M.; George, J. F. (2016): Toward the development of a big data analytics capability. *Information & Management*, vol. 53, no. 8, pp. 1049-1064.

Jiang, W.; Song, H.; Dai, Y. (2005): Real-time intrusion detection for high-speed networks. *Computers & Security*, vol. 24, no. 4, pp. 287-294.

Karger, D. R.; Lehman, E.; Leighton, T.; Panigrahy, R.; Levine, M. et al. (1997): Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. *Proceedings of the International Symposium on Theory of Computing (ACM STOC)*, pp. 654-663.

Konstantinos, X.; Ioannis, C.; Spiros, A. (2006): An active splitter architecture for intrusion detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 1, pp. 31-44.

Kruegel, C.; Valeur, F.; Vigna, G.; Kemmerer, R. (2002): Stateful intrusion detection for high-speed networks. *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pp. 285-294.

Lai, H.; Cai, S.; Huang, H. (2004): A parallel intrusion detection system for high-speed networks. *Proceedings of Applied Cryptography and Network Security, Second International Conference*, pp. 439-451.

Liu, Y. H.; Xin, T. D.; Gang, Y. X.; Jian, W. (2008): Large-Scale network intrusion detection algorithm based on distributed learning. *Journal of Software*, vol. 19, no. 4, pp. 993-1003.

Rosas, C.; Sikora, A.; Jorba, J. (2014): Dynamic tuning of the workload partition factor and the resource utilization in data-intensive applications. *Future Generation Computer Systems*, vol. 37, no. 6, pp. 162-177.

Schaelicke, L.; Wheeler, K.; Freeland, C. (2005): A scalable network intrusion detection loadbalancer. *Proceedings of the 2nd Conference on computing Frontiers*, pp. 315-322.

Soklic, M. E. (2002): Simulation of load balancing algorithms: A comparative study. *ACM SIGCSE Bulletin*, vol. 34, no. 4, pp. 138-141.

Wang, X. T.; Shen, D. R.; Mei, B. (2016): An efficient algorithm for distributed outlier detection. *Chinese Journal of Computers*, vol. 39, no. 1, pp. 36-51.

Wang, Z.; Chen, Q.; Li, Z. H.; Pan, W.; You, L. (2016): An incremental partitioning strategy for data balance on mapreduce. *Chinese Journal of Computers*, vol. 39, no. 1, pp. 19-35.