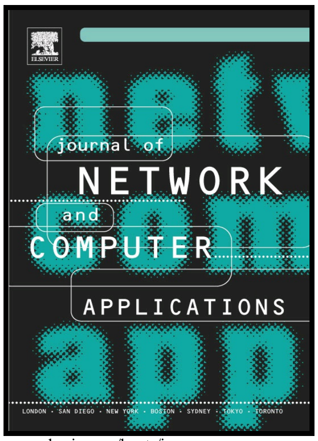
Efficient and Robust Dynamic Network Traffic Partitioning Based on Flow Tables

Bing Xiong, Kun Yang, Jinyuan Zhao, Keqin Li


www.elsevier.com/locate/jnca

PII: S1084-8045(16)30063-7
DOI: http://dx.doi.org/10.1016/j.jnca.2016.04.013
Reference: YJNCA1627

To appear in: *Journal of Network and Computer Applications*

Received date: 23 November 2014
Revised date: 30 October 2015
Accepted date: 16 April 2016

Efficient and Robust Dynamic Network Traffic Partitioning Based on Flow Tables

Bing Xiong[a,b,1,*], Kun Yang[c], Jinyuan Zhao[d], Keqin Li[e]

[a]*Hunan Provincial Key Laboratory of Intelligent Processing of Big Data on Transportation, Changsha University of Science and Technology, Changsha 410114, China*
[b]*School of Computer and Communication Engineering, Changsha University of Science and Technology, Changsha 410114, China*
[c]*School of Computer Science and Electronic Engineering, University of Essex, Wivenhoe Park, Colchester CO43SQ, UK*
[d]*School of Software, Central South University, Changsha 410075, China*
[e]*Department of Computer Science, State University of New York at New Paltz, New York 12561, USA*

Abstract

The continual growth of network traffic rates results in heavy traffic processing overloads, and a typical solution is to partition traffic into multiple network processors for parallel processing especially in software-defined networks. This paper proposes an efficient and robust network traffic partitioning scheme called DTP-ICM, which dynamically distributes packet traffic with flow granularity, and assigns a new flow to a network processor with the lightest traffic load level. The scheme is first improved by building an initial connection separation mechanism to isolate false TCP connections, which results in the initial connection table (ICT) and the established connection table (ECT). The move-to-front heuristic is applied to the ECT table and the UDP flow table to accelerate their lookups and to the ICT table for fast timeout scanning. Besides, the ICT table is optimized with counting bloom filters to defend against SYN flooding attacks. Finally, we evaluate the performance of our DTP-ICM scheme with real network traffic traces by experiments. Experimental results indicate that the DTP-ICM scheme outperforms the traditional ones in terms of traffic load balance, packet

*Corresponding author
Email address: `xiongbing@csust.edu.cn,xiongbing.csust@qq.com` (Bing Xiong)
[1]Phn: +86-18773116229, Fax:+86-0731-85258462

distribution efficiency and robustness against malicious behaviors.

Keywords: software-defined networks, dynamic traffic partitioning, SYN flooding attacks, initial connection separation, counting bloom filters, move-to-front heuristic

2015 MSC: 00-01, 99-00

1. Introduction

The continual growth of network bandwidth leads to the fact that a single network device cannot real-timely process large-scale packet traffic, especially in high-speed networks with bandwidth 1Gps and above. A typical solution is load balancing, which partitions heavy network traffic into many parts and forwards them to multiple network processors for parallel processing. Network traffic partitioning is a key to load balancing, and has been extensively applied in network applications, such as stateful firewalling[1], intrusion detection[2][3], traffic measurement[4], high-speed packet switching[5], and content delivery[6][7]. In particular, emerging network paradigms such as software-defined networking[8][9] make load balancing much easier to be deployed without changing substrate devices by using OpenFlow[10][11].

Network traffic partitioning in these applications is generally required to satisfy the following properties. Firstly, network traffic must be partitioned with flow granularity to support packet processing at session levels above the network layer[12][13]. In particular, all packets within a flow must be assigned to an identical network processor. For convenience, the terms flow and session will be used interchangeably in this paper, and both refer to packet traffic between two communication endpoints in networks, particularly called connection for TCP. Secondly, network traffic should be dynamically partitioned in terms of the processing capacities of network processors to achieve good load balance. Thirdly, each packet must be distributed at fast speeds to real-timely process large-scale network traffic even in the presence of malicious attacks.

There exist many network traffic partitioning schemes, which can be classified into three types: direct hashing[14][15], hash space division[18] and flow table based dynamic schemes[21][22][23][24]. The direct hashing schemes map each packet to a network processor by a hash function, which is only suitable for network processors with identical configurations. The hash space division schemes divide the hash space into many intervals assigned to network processors in terms of their processing capacities. This type adapts to heterogeneous network processors, but is still hard to achieve good load balance due to no consideration of non-uniform distribution of network traffic. The flow table based dynamic schemes maintain a hash table of simultaneous flows, and dynamically distribute packet traffic with flow granularity to multiple network processors. This type keeps flow integrity during packet distribution and balances traffic load well among network processors. However, their packet distribution performance becomes a great challenge when they are manipulated in high-speed networks especially in the presence of malicious behaviors.

Many literatures to date have achieved a deep insight into the load balancing capacity of dynamic traffic partitioning[21][22][23][24]. Unfortunately, there is few research work to optimize the packet distribution performance to the best of our knowledge. Chen et al.[17] dynamically remapped the flow bundle with the least number of TCP flows to the lightest loaded processing unit when traffic load became imbalanced between processing units. Sun et al.[18] mapped each incoming packet by hashing its key header fields to an interval allocated for an processing unit, and the interval was fine-tuned in terms of processing capacity and current load of all processing units. These methods achieve real-time traffic load balance, but will lead to frequent load migration due to bursty network traffic. Shi et al.[19] classified Internet flows into two categories: the aggressive and the normal, and applied dynamic scheduling policies to the aggressive flows. Kencl et al.[20] designed a feedback control mechanism to prevent processor overload, and provided an adaptive extension of the highest random weight (HRW) scheme to cope with biased traffic patterns. These schemes achieve good load balancing and high processor utilization, but do not help to stand against malicious behaviors such as SYN flooding attacks.

This paper focuses on how to optimize the packet distribution performance of dynamic traffic partitioning scheme. The optimization is investigated with the following methodology. We first give the framework of dynamic network traffic partitioning scheme, and improve its flow table management by splitting the hash table of TCP connections. Then, all flow tables are optimized by employing heuristic methods to speed up their lookups and timeout scanning. As a key step, we resist against SYN flooding attacks by applying counting bloom filters to the lookups of the initial connection table. Finally, we give the algorithm implementation of our proposed traffic partitioning scheme, and evaluate its packet distribution performance with real network traffic traces.

With the above methodology, we aims to achieve the following conclusions as the main contributions of this paper: (a) Giving the framework of dynamic traffic partitioning scheme, which reaches real-time traffic load balance with the guarantee of flow granularity; (b) Building the initial connection separation mechanism to improve flow table management performance by dividing the TCP connection table into the initial connection table (ICT) and the established connection table (ECT); (c) Optimizing the ICT table lookups of TCP SYN packets by employing counting bloom filters to defend against SYN flooding attacks; (d) Applying the move-to-front heuristic to accelerate the lookups of the ECT table and the UDP flow table, and the timeout scanning of the ICT table.

The rest of this paper is organized as follows. In Section 2, we introduce the related work. Section 3 describes the fundamental principle of our proposed traffic partitioning scheme DTP-ICM, whose flow table management is improved by building the initial connection separation mechanism. In Section 4, we optimize the two resultant TCP connection tables and the UDP flow table by employing the move-to-front heuristic and counting bloom filters. Section 5 describes the algorithm implementation of the DTP-ICM scheme, whose performance is evaluated with physical network traffic traces in Section 6. Section 7 concludes the paper.

2. Related Work

In the last decade, there have been many literatures on traffic load balancing in various network applications. Most of their work focused on the load balancing capacity of network traffic partitioning, but rarely contributed to the packet distribution performance, especially robustness against malicious behaviors.

Cao et al.[15] designed a table-based hashing scheme for the scenario of several network processors with different capacities in Internet traffic load balancing. The scheme splits a traffic stream into multiple bins mapped into outgoing links based on an allocation table. However, the algorithm has poor adaptability of load balance, and it has been pointed out that hashing alone is not able to balance network traffic workload[19]. Lai et al.[16] proposed a traffic partitioning algorithm for parallel intrusion detection systems. They employed hash table to maintain simultaneous TCP connections, and partitioned network traffic in virtue of TCP connection state. The algorithm provides better load balancing capacity than direct hashing schemes. However, it does not consider TCP state accurately, and ruins the integrity of connection context during packet scheduling.

Chen et al.[17] presented a session-oriented adaptive load balancing algorithm based on IP header multi-field classification. The algorithm dynamically adjusts flow bundles to guarantee session integrity when traffic load becomes imbalanced between processing units. To keep dynamic balance between processing units, they remapped the flow bundle with the least number of TCP flows to the lightest loaded processing unit. Sun et al.[18] provided a novel load balancing algorithm for parallel intrusion detection systems. The algorithm maps each incoming packet by hashing its key header fields to an interval allocated for an IDS sensor. The interval is fine-tuned in terms of processing capacity and current load of all IDS sensors to achieve real-time balance of traffic distribution. However, these algorithms will produce frequent flow adjustment because of bursty network traffic, which leads to a lot of load migration between the sensors.

Targeting load balancing between forwarding engines in Internet routers, Shi et al.[19] classified Internet flows into two categories: the aggressive and the normal, and applied dynamic scheduling policies to the aggressive flows to achieve both load balancing and efficient system resource utilization. Kencl et al.[20] presented an adaptive load balancing scheme for load sharing among multiple network processors within a router. They designed a feedback control mechanism to prevent processor overload, and provided an adaptive extension of the highest random weight (HRW) scheme to cope with biased traffic patterns. The scheme achieves significant improvement in processor utilization, and minimizes the probability of flow reordering by exploiting the minimal disruption property of the adjustment of the packet-to-processor mapping.

Li et al.[21] proposed an application-based dynamic-least-load-first algorithm for high-speed network intrusion detection systems. They real-timely maintained a hash table of all assigned sessions and dynamically scheduled new sessions in terms of current load levels of all intrusion analyzers. Jiang et al.[22][23] discussed a flow-based dynamic traffic partitioning algorithm for intrusion detection systems in high-speed networks. The algorithm divides packet stream with flow granularity and forwards a packet of a new session to the detection engine with the least load currently. Xiong et al.[24] dynamically maintains a hash table of concurrent sessions, and assigns a session to a network processor with the lightest load level when the session appears. These algorithms achieve good effect of traffic load balancing, but their packet distribution performance is not adequately considered.

To achieve better load balancing performance in high-speed networks, this paper proposes an efficient and robust dynamic network traffic partitioning scheme. In the scheme, we first build the initial connection separation mechanism to isolate false TCP connections especially induced by SYN flooding attacks, which results in the initial connection table (ICT) and the established connection table (ECT). As a further step, the move-to-front heuristic is applied to the ECT table and the UDP table to accelerate their lookups and to the ICT table for fast timeout scanning. Moreover, the ICT table is optimized by employing counting bloom filters to defend against the attacks. By this way, we aim to boost the packet distribution performance of dynamic traffic partitioning scheme.

3. Dynamic Traffic Partitioning Scheme

This section describes the framework of our dynamic traffic partitioning scheme and divides its TCP connection table into two tables by building an initial connection separation mechanism for good packet distribution performance.

3.1. Framework

Many network applications relating to stateful packet processing usually employ load balancing technology to cope with massive packet traffic. One of their key problems is traffic partitioning scheme. The traffic partitioning scheme in stateful packet processing usually needs to satisfy the following properties[12]: (a) flow granularity. Network traffic must be partitioned with flow granularity to support packet processing at semantic levels; (b) dynamic balance. Network traffic should be dynamically partitioned to achieve real-time load balance among all network processors; (c) fast distribution. Each arrived packet must be distributed at fast speeds to real-timely process large-scale network traffic; (d) good robustness. The traffic partitioning scheme should perform soundly even in the presence of malicious behaviors.

This paper proposes a novel dynamic traffic partitioning scheme based on flow tables, which aims to meet the above properties. We first give the basic idea of our traffic partitioning scheme as follows. We maintain simultaneous flows of each protocol above IP in a separate hash table. Each flow in a table is associated with a network processor assigned at its appearance. In particular, we assign a new flow to a network processor with the lightest traffic load level, and all packets within the flow are forwarded to the network processor. To determine the lightest loaded network processor, we build the load state table to real-timely maintain the traffic load states of all network processors.

According to the above basic idea, we further illustrate the fundamental principle of our traffic partitioning scheme in *Fig.1*. Suppose there are N network processors, and the traffic partitioner maintains a flow table with the length H. As for a continuous stream of network packets $(p_1, p_2, \cdots, p_i, \cdots)$, the traffic partitioner forwards them in terms of their arrival orders. With regard to a specific packet p_i, we get its flow identifier from its header fields and map the identifier to an entry of the flow table $FT_h(0 \leq h \leq H-1)$. Then, we look up the flow list in the entry FT_h for a match $f_{h,j}$. If the lookup succeeds, we directly forward the packet to the network processor $NP_n(0 \leq n \leq N-1)$, whose number n is kept in the flow $f_{h,j}$. Otherwise, the packet p_i is supposed to belong to a new flow. In this case, the flow is assigned to the lightest-loaded network processor $NP_m(0 \leq m \leq N-1)$ in terms of the load state table LST, which keeps real-time load information of each network processor. Meanwhile, we register the flow and its assigned network processor number m in the flow table FT. Finally, the packet p_i is forwarded to the network processor NP_m.

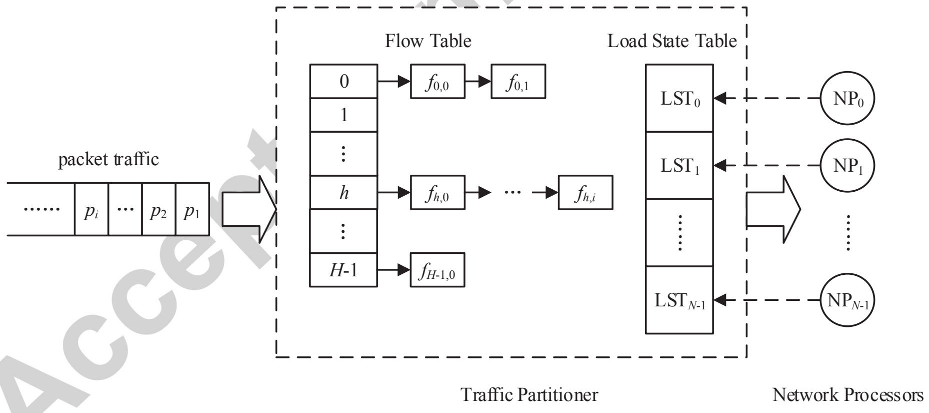


Figure 1: The fundamental principle of our traffic partitioning scheme.

Dynamic traffic partitioning at the micro level is to distribute each incoming packet to a network processor. As seen from the above fundamental principle, the essential operation of packet distribution is the flow table lookup, which chiefly depends on flow table management. A typical way of flow table management is to maintain all simultaneous flows of each protocol in the IP header with a single hash table, called single flow table (SFT) scheme. This scheme works well for 100Mbps networks or below, since their bandwidth limits the flow tables to be within an acceptable size. However, the flow tables will be much larger in high-speed networks with bandwidth 1Gbps and above, where there are up to hundreds of thousands of simultaneous flows[25]. Fortunately, their lookup overheads can still be limited by selecting a biggish flow table length. Suppose there are up to 10^6 (about 2^{20}) simultaneous flows for a transport-layer protocol such as TCP or UDP, its flow table will have the load factor only 16 if its table length is configured as 2^{16}, and its lookup overheads will be apparently acceptable with uniform hashing.

Unfortunately, the SFT scheme can no longer perform soundly in the presence of malicious behaviors such as SYN flooding attacks. Such attacks subvert flow table management by inducing an avalanche of false TCP connections into the TCP flow table. Despite these false connections can be eliminated in a short time by timeout mechanism, the TCP flow table still has to accommodate a large number of unexpired connections besides of massive normal connections. This results in its heavy lookup overheads with the addition of highly intensive packets in high-speed networks. Besides, the huge TCP flow table has to be frequently traversed to clear out expired connections due to the presence of malicious attacks. This leads to its additional heavy timeout scanning overheads. In summary, the TCP flow table will face great challenges under network attacks on the account of its too heavy lookup and timeout scanning overheads. Therefore, it is necessary to optimize flow table management for better packet distribution performance.

3.2. TCP Connection Table Division

Malicious behaviors such as SYN flooding attacks have destructive impact on flow table management especially TCP connection table. The attackers bring a great amount of false connections into the table by intensively sending spoofed SYN packets, and make the table expand rapidly and its operation overheads rise sharply. Note that all false connections induced by such attacks will not complete three-way handshake to establish a TCP connection. In virtue of this feature, we build an initial connection separation (ICS) mechanism to isolate false connections from normal connections. The essential concept of the ICS mechanism is to separate initial connections including all false connections from the TCP connection table and manipulate them separately. As for its implementation level, we run two TCP connection tables: (a) the initial connection table (ICT), whose connections have been initiated but not yet completed the three-way handshake; (b) the established connection table (ECT), whose connections have been established.

According to the above scheme, we illustrate the lifetime of a TCP connection in our connection management as *Fig.2*. When a new connection appears, its record will be generated into the ICT table. Once the connection is established, the record will be immediately transferred to the ECT table. When the connection is terminated, we will delete the record from the ECT table. Subsequently, a TCP connection is manipulated as follows on the arrival of a packets. When a SYN packet appears, we create its connection and insert it into the ICT table. If the third handshake ACK arrives, we take the connection out of the ICT table and put it in the ECT table. Once the last packet ACK arrives, we remove the connection from the ECT table. In addition, whenever a RST packet resetting a connection arrives, we immediately delete its connection from the ICT table or the ECT table.

The above ICS mechanism results in controllable size of both the ICT table and the ECT table. The ICT table contains a small amount of connections under normal conditions, due to the fact that 93% and 99% of TCP connections respectively take less than 1 second and 4 second to complete three-way handshake[26][27]. When network attacks appear, we strive to limit the inflating ICT table by setting a small timeout interval (e.g. 1 second), which eliminates false connections as early as possible. Then the ICT table maintains a controllable number of false connections produced during the interval, besides of normal initial connections. Fortunately, the ECT table is immune to malicious

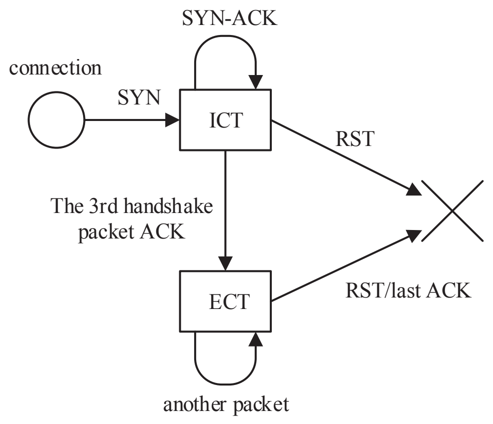


Figure 2: The lifecycle of a connection in the ICS mechanism.

behavior and always kept within an acceptable size even in high-speed networks. When a TCP packet arrives, its connection resides in either the ICT table or the ECT table in most cases. Therefore, we only need to look up one of the two connection table generally, and the lookup overheads will be greatly reduced. In summary, the ICS mechanism divides single oversize TCP connection table into two controllable tables, and makes network attacks much easier to cope with.

4. Flow Table Optimization

This section optimizes flow tables by applying the move-to-front (MTF) heuristic and counting bloom filters for high lookup efficiency and good robustness against malicious behaviors.

4.1. Table Operation Analysis

As seen from the previous section, our dynamic traffic partitioning scheme runs the ICT and ECT tables for TCP and a single flow table for each other transport-layer protocol typically UDP. There are two operation types for a flow table, lookups on packet arrivals and periodic timeout scanning. Then we analyze the properties of both operations on these flow tables.

1) The ICT table

The ICT table keeps a relatively small scale even in high-speed networks under normal conditions, and performs lookups and traversals at fast speeds. Thus, the ICT table holds low lookup overheads as it is looked up only for the three-way handshake packets within a TCP connection. Meanwhile, the table also has cheap scanning overheads due to low frequency of timeout scanning under normal conditions.

When network attacks occur, the ICT table is sharply expanded by an avalanche of false connections, which leads to its heavy lookup and traversal overheads. On the one hand, the ICT table is looked up frequently due to the emergence of a large number of falsified packets. On the other hand, we need to go through the ICT table for all SYN packets including falsified ones. These factors result in a significant increase in the overall lookup overheads. Since the dominant SYN packets will not find out a connection in the ICT table, we can try to find a way to give their result soon without the table lookup. If the way is successfully found, it will decrease the lookup overheads of the ICT table to a remarkable extent, and alleviate the damage of network attacks to the ICT table.

As for timeout scanning, the ICT table has to be scanned at high frequency in the presence of network attacks. This leads to a great raise in its timeout scanning overheads. However, it will be much more convenient to clear out false connections, if each bucket in the ICT table is sorted by the last access time of its connections. In such case, all expired connections converge on the rear of the connection list in each bucket, and can be directly deleted from the tail of each connection list, other than going through the table to find them. If we find out an optimization method to achieve the effect above, it will apparently decrease the timeout scanning overheads of the ICT table.

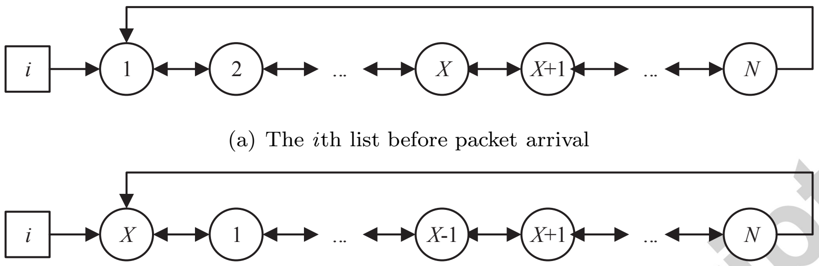
2) The ECT table and the UDP flow table

The ECT table and the UDP flow table always keep a large size in high-speed networks, and perform lookups and traversals at high cost. Fortunately, the ECT table is immune to malicious behaviors such as SYN flooding attacks, and there are rare attacks for UDP to inflate its flow table. This leads both tables to be scanned at low frequency and results in their moderate timeout scanning overheads. As for their lookup operations, we need to search the UDP flow table and the ECT table respectively on the arrival of a UDP packet and each TCP packet except the three-way handshake ones. So both tables will be looked up frequently due to intensive packet traffic in high-speed networks. These factors lead to their heavy lookup overheads, which need to be optimized. Network traffic investigations indicate that packet traffic exhibit locality phenomenon in packet switched networks. A common observation is that 10% of the hosts account for 90% of the traffic[28][29]. The locality becomes increasingly evident with the widespread application of peer-to-peer technologies[30][31]. The locality is manifested as packets arrive in group with regard to a flow, and means for the flow table lookups that a flow in the table will be successively referenced in a short time. In other words, if the referred flow is adjusted to the head of the table after every lookup, the lookup for subsequent packets will be much faster to locate their flows. Therefore, we exploit the locality by applying the move-to-front heuristic to the ECT table and the UDP flow table for better lookup performance.

4.2. Move-to-front Heuristic

As seen from the above analysis of flow table operations, we can apply the move-to-front (MTF) heuristic[32] to optimize the lookups of the ECT table and the UDP flow table and the timeout scanning of the ICT table. In particular, for each time that the lookup of a flow table hits a flow, the flow is shifted to the front of its located list in the table. *Fig.3* illustrates the working principle of a flow list applying the move-to-front heuristic. *Fig.3(a)* gives a schematic of the ith list just before the arrival of a packet on flow *"X"*, and *Fig.3(b)* gives a schematic of the list just after the arrival. Note that flow *"X"* has been pulled to the front of the list. *Table.1* demonstrates the pseudo-code for the operations on a flow table applying the move-to-front heuristic.

The MTF heuristic results in a significant boost in the lookup performance of the ECT table and the UDP flow table. Owing to network traffic locality, packet



(a) The ith list before packet arrival

(b) The ith list after packet arrival

Figure 3: The principle of a flow list applying the move-to-front heuristic.

Table 1: The operations on a flow table applying the move-to-front heuristic

The search (including deletion) of a flow table	The insertion of a flow table	The timeout scanning of a flow table
FT_search(FlowTable *table*, FlowIdentifier *key*)	FT_insert(FlowTable *table*, flow **p*)	FT_timeout(FlowTable *table*, int *interval*)
1. $pos \leftarrow$ Hash(key)	1. $pos \leftarrow$ Hash($p \rightarrow key$)	1. get the current time t_cur
2. $list \leftarrow table[pos].head$	2. $list \leftarrow table[pos].head$	2. **for** $i \leftarrow 1, table.len$ **do**
3. **for** $p \leftarrow list.next, \&list$ **do**	3. $p \rightarrow next \leftarrow list.next$	3. **while** TRUE **do**
4. **if** $p.key = key$, **then**	4. $p \rightarrow prior \leftarrow \&list$	4. $p \leftarrow table[i].head.prior$
5. $p \rightarrow prior \rightarrow next \leftarrow p \rightarrow next$	5. $list.next \rightarrow prior \leftarrow p$	5. **if** t_cur-$p \rightarrow time > interval$, **then**
6. $p \rightarrow next \rightarrow prior \leftarrow p \rightarrow prior$	6. $list.next \leftarrow p$	6. $p \rightarrow prior \rightarrow next \leftarrow p \rightarrow next$
7. **return** p	7. **return** 1	7. $p \rightarrow next \rightarrow prior \leftarrow p \rightarrow prior$
8. **return** 0		8. $keyset \leftarrow keyset \cup p \rightarrow key$
		9. delete p
		10. **else break**
		11.**return** $keyset$

flows can be classified into active flows and inactive flows, which account for a large and small percentage of packet traffic respectively. All frequently referred flows are shifted to the front of the table after a series of the table lookups. These cases result in two consequences: (a) a slight increase in the search length of inactive flow with a handful of individual packets; (b) a substantial decrease in the search length of active flow with a mass of successive packets. In conclusion, the MTF heuristic will bring a great decrease to the lookup overheads of the ECT table and the UDP flow table.

Furthermore, the MTF heuristic leads each hash list of the ICT table to be sorted by the last accessing time of each connection in descending order, and all expired connections to converge on the rear of each hash list. Thus we can easily execute timeout scanning by deleting each expired connection from the tail of all hash lists. With no need to traverse each hash list to get all expired connections, the timeout scanning greatly speeds up especially at the appearance of enormous falsified connections under SYN flooding attacks. In conclusion, the MTF heuristic will result in an apparent reduce in the timeout scanning overheads of the ICT table, which helps to defend against network attacks.

4.3. Counting Bloom Filters

The ICT table is sharply expanded and its lookup overheads are rapidly rising in the presence of SYN flooding attacks. This is attributed to the fact that the dominant SYN packet looks up the table for its connection in failure. Since the failed lookups are already known for most of packets, we can employ counting bloom filters to directly give their lookup results without searching the table.

A bloom filter[33] is a simple space-efficient data structure for representing a set in order to support membership queries. A counting bloom filter[34] generalizes a bloom filter data structure so as to allow that the set can be changing dynamically via insertions and deletions. *Fig.4* illustrates the working principle of the ICT table employing a counting bloom filter. It is described by an array

A of m counters (with several bits), initialized to 0. And It uses k independent hash functions $h_1, h_2, \cdots, h_k$, each with range $\{1, \cdots, m\}$. It has a set C of n elements $c_1, c_2, \cdots, c_n$, i.e., connection identifiers in the ICT table.

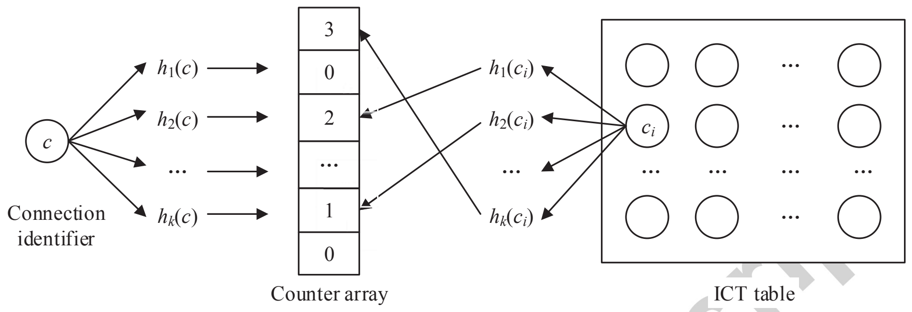


Figure 4: The principle of the ICT table employing a counting bloom filter.

If an element c is to be inserted into the set C, the counters $A[h_i(c)]$ $(1{\leq}i{\leq}k)$ at position $h_1(c), h_2(c), \cdots, h_k(c)$ in A are incremented by 1 accordingly. If an element c is to be deleted from the set C, the counters $A[h_i(c)]$ at position $h_1(c), h_2(c), , h_k(c)$ in A are decremented by 1 accordingly. If we want to query for an element c, we check all the value of the counters $A[hi(c)]$ $(1{\leq}i{\leq}k)$. If any of them is 0, then certainly c is not in the set C. Otherwise, we conjecture that the element c is in the set C, although there is a certain probability that we are wrong. This is called a "false positive". In such case, we still need to search the ICT table for an exact result. *Table.2* summarizes the above discussion regarding the operations on the ICT table employing a counting bloom filter.

The probability for a false positive error is dependent on the parameters k, m/n. For the counting bloom filter, after n connections were inserted at random into the counter array of size m, the probability that a particular counter is 0 is exactly $(1 - 1/m)^{kn}$. Hence the probability of a false positive in this situation is[34][35]

$$\left(1 - (1 - 1/m)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k. \quad (1)$$

The right-hand expression in (1) is minimized for $k = ln2 * m/n$, in which

Table 2: The operations on the ICT table employing a counting bloom filter

The search (including deletion) of the ICT table	The insertion of the ICT table	The timeout scanning of the ICT table
ICT_CBF_search(FlowIdentifier key)	ICT_CBF_insert(connection $*p$)	ICT_CBF_timeout(int $interval$)
1. **for** $i \leftarrow 1,\ k$ **do**	1. FT_insert($ICT,\ p$)	1. $keyset \leftarrow$ FT_timeout($ICT,\ interval$)
2. $pos \leftarrow H_i(key)$	2. **for** $i \leftarrow 1,\ k$ **do**	2. **while** $keyset \neq \emptyset$ **do**
3. **if** $CBF[pos] \neq 0$, **then**	3. $pos \leftarrow H_i(p \rightarrow key)$	3. get a key from $keyset$
4. **break**	4. $CBF[pos] \leftarrow CBF[pos] + 1$	4. **for** $i \leftarrow 1,\ k$ **do**
5. **if** $i \leq k$, **then**	5. **return** 1	5. $pos \leftarrow H_i(key)$
6. $p \leftarrow$ FT_search($ICT,\ key$)		6. $CBF[pos] \leftarrow CBF[pos] - 1$
7. **if** $p \neq NULL$, **then**		7. **else break**
8. **for** $i \leftarrow 1,\ k$ **do**		8. **return** 1
9. $pos \leftarrow H_i(p \rightarrow key)$		
10. $CBF[pos] \leftarrow CBF[pos] - 1$		
11. **return** p		
12.**return** 0		

case the error rate is $(1/2)^k = (0.6185)^{m/n}$. For example, the false positive error rate is slightly larger than 2.15%, when $m/n = 8$, $k = 6$. The false positive error rate is only 0.314%, when $m/n = 12$, $k = 8$.

For the counting bloom filter, it is also important to know how large the memory of the counters can become. In order to determine a good counter size, we consider this situation: after inserting n connections with k hash functions into a counter array of size m, the probability that the jth counter is greater or equal i is

$$Pr(A[j] \geq i) \leq \Big(\frac{enk}{im}\Big)^i, (1 \leq j \leq m). \tag{2}$$

As mentioned above, we can optimize the false positive rate with $k = ln2 * m/n$, so we assume that we restrict ourselves to $k = ln2 * m/n$, then

$$Pr(\max_j A[j] \geq i) \leq m\Big(\frac{eln2}{i}\Big)^i. \tag{3}$$

If we set 4 bits per counter, the counter will overflow if and only if some counter reaches the value 16. From the above, we know that

$$Pr(\max_j A[j] \geq 16) \leq m\Big(\frac{eln2}{16}\Big)^{16} \approx 1.37 \times 10^{-15} \times m. \tag{4}$$

Thus, 4 bits per counter will suffice for most applications. In conclusion, the counting bloom filter is highly effective. With the aid of the counting bloom filter, we can directly tell the lookup results of the ICT table without need to search any flow table for most SYN packets. As a consequence, the lookup overheads of the ICT table will be greatly controlled even under SYN flooding attacks. By this way, we significantly cut down the damage of malicious behaviors on the lookup performance of flow tables.

5. Packet Distribution Algorithm

Our proposed network traffic partitioning scheme above is called DTP-ICM, short for dynamic traffic partitioning (DTP) with the initial connection separation (ICS) mechanism further improved by counting bloom filters (CBF) and the move-to-front (MTF) heuristic. This section describes the algorithm implementation of the DTP-ICM scheme based on TCP packet classification.

5.1. TCP Packet Classification

Dynamic traffic partitioning at the micro level is packet distribution performed by four basic steps: (a) parsing the packet to get its key fields and calculating its flow identifier; (b) looking up the corresponding flow table for a match by the flow identifier; (c) distributing the packet to the network processor in the matched flow or else with the lowest traffic load level; (d) operating (creating, inserting, shifting or deleting) the flow in the flow table after its update. An essential work of packet distribution is to look up the respective flow table. There is no doubt for a non-TCP packet since we maintain a respective flow table for each protocol above IP header except TCP. As for a TCP packet, we must determine which table is to be looked up as there are two TCP connection tables in our scheme. If the lookup fails, we still need to decide whether the other table should be further searched. To clarify these problems, we classify TCP packets into five types as follows.

1) Packets with SYN flag. A packet with SYN flag initiates its half of a TCP connection and appears during connection establishment phase. Thus its connection must reside in the ICT table if being exists.

2) Packets with FIN flag. A packet with FIN flag tears down its half of TCP connection and appears during connection termination phase. So its connection must be reserved in the ECT table.

3) Packets with RST flag. A packet with RST flag is sent whenever a segment arrives which apparently is not intended for the current connection. The packet is generated in 3 typical cases[36]: (a) a connection request is delivered to a non-existent port. This situation happens at the beginning of connection establishment, and its connection must reside in the ICT table; (b) a connection endpoint abort its connection in response to an unacceptable segment or a termination command from its application. These causes generally arises at the end of data transfer, and its connection should stay in the ECT table; (c) one end detects a half-open connection, whose other end has closed or aborted the connection without its knowledge. This case occurs at data transfer phase, and its connection must be saved in the ECT table.

4) Pure ACK. A pure ACK packet does not carry any payload and key TCP flags, i.e., SYN, RST and FIN. The packet has three possibilities: (a) the 3rd handshake during connection establishment, whose connection lies in the ICT table; (b) an acknowledgement of some transmitted segment during data transfer phase, whose connection is located in the ECT table; (c) an acknowledgement of connection tear-down request, whose connection also resides in the ECT table. In summary, its connection is more likely to stay in the ECT table. Therefore, we will first look up the ECT table on its arrival. If the lookup fails, we continue to search the ICT table for a match.

5) Impure ACK. An impure ACK packet carries payload but no key TCP flags (SYN, RST and FIN). The packet usually turns up during data transfer phase, and its connection must be maintained in the ECT table.

5.2. Algorithm Description

Upon receiving a packet, we first parse it to get its key fields with respect to protocol header format at each layer, and calculate its flow identifier with its source/destination IP addresses and port numbers defined below. The port numbers are set as zero for any protocol in which header there is no port number field, such as ICMP. Then we operate the flow table corresponding to the protocol in its IP header. As for a non-TCP packet, we directly search the respective flow table for a match. If we fail to get a match, we create a new flow and assign it to a network processor with the lightest load degree in terms of the load state table. Then, we distribute the packet to the network processor, update the flow with the packet, and insert it to the respective flow table.

Definition 5.1 (Flow Endpoint Identifier (FEI)). *An endpoint of a network flow can be identified as a 2-tuple $FEI(IP, PT)$, where IP and PT respectively represent IP address at the network layer and port number at the transport layer.*

Definition 5.2 (The relationship of two FEIs). *Suppose there are two flow endpoints, $FEI_1(IP, PT)$ and $FEI_2(IP, PT)$. Considering IP and PT in each FEI as a 32-bit and 16-bit integer respectively, we define the relationship of the two FEIs as $FEI_1 < FEI_2$, iff (a) $FEI_1.IP < FEI_2.IP$ or (b) $FEI_1.IP = FEI_2.IP$ and $FEI_1.PT < FEI_2.PT$.*

Definition 5.3 (Flow Identifier (FID)). *There are two opposite endpoints in a flow. Suppose FEI_S and FEI_B is respectively the smaller and the bigger of them in accordance with Denifition 5.2, the flow can be identified as 2-tuple $FID(FEI_S, FEI_B)$.*

The distribution of a TCP packet is performed in terms of the packet classification above. As for a SYN packet, we search the ICT table with counting bloom filters. If the search fails, the packet is confirmed to initiate a new connection. Then we create a new connection for it and assign it to a network processor with the lightest traffic load degree. As for a SYN/ACK or RST/ACK packet, we directly look up the ICT table for its connection. The ECT table is searched for any other packet. If the search fails and the packet is a pure ACK, the packet is probably the third handshake and we continue to directly look up the ICT table. Up to now, we are supposed to get a connection. Then, we distribute the packet to the network processor in the connection and update the connection with the packet. Finally, the connection is manipulated in terms of its state. In particular, the connection will be directly deleted if it reaches the termination state. Otherwise, we insert it into the ICT table if it is not yet established and the ECT table for any other case. *Table.3* summarizes the above discussion regarding the packet distribution of our proposed DTP-ICM scheme.

In the meanwhile, the timeout scanning is manipulated on each flow table to clear out expired flows in time. In particular, the removal of expired flows from the ICT table triggers the update of its counting bloom filter. The timeout interval of the ICT table is set as 4 second under normal conditions due to the fact that 99% of TCP connections take less than 4s to complete three-way handshake. When SYN flooding attacks occur, the interval is tuned to 1 second as 93% of TCP connections can also be established. As for other flow tables including the ECT table and the UDP session table, their timeout intervals are configured as 60 second. *Table.4* demonstrates the pseudo-code for the flow table timeout of our proposed DTP-ICM scheme.

5.3. Algorithm Complexity Analysis

As seen from the algorithm description above, the performance of our traffic partitioning scheme mainly depends on the flow table lookups. According to network traffic locality, packet traffic within a flow can be modeled as a number of packet trains, each of which consists of several packets in either direction. A flow is active when one of its trains is running and inactive if a train has passed away and the next train has not come yet. Owing to the move-to-front heuristic, all active flows will be shifted to the front of their located lists in the flow table while all inactive flows will be pushed back to the rear. As a consequence, each list in the flow table can be divided into two parts, the front active flow zone

Table 3: The packet distribution of the DTP-ICM scheme

Pseudo-code for the distribution of a packet

```
Alogrithm 1 PacketDistribution(Packet $p$)
1. Parse the received packet $p$ to get its key fields, including $ip_{src}, ip_{dst}, port_{src}, port_{dst}, proto$
2. $p.FID \leftarrow \{ip_{src}, ip_{dst}, port_{src}, port_{dst}, proto\}$
3. if $p.proto \neq TCP$, then
4.     $f \leftarrow$ FT_search($FT_{p.proto}$, $p.FID$)
5.     if $f = NULL$, then
6.         $k \leftarrow \min_i\{LoadLevel(NP_i)\}$
7.         $f \leftarrow$ NewFlow($p$, $k$)
8.     Forward the packet $p$ to the network processor in the flow f, and update the flow $f$ with the packet
$p$
9.     FT_insert($FT_{p.proto}$, $f$)
10.    return 1
11.else return DTP-ICM_TCP($p$)
12.if $p$ is a SYN, then
13.    $c \leftarrow$ ICT_CBF_search($p.FID$)
14.    if $c = NULL$, then
15.        $k \leftarrow \min_i\{LoadLevel(NP_i)\}$
16.        $c \leftarrow$ NewFlow($p$, $k$)
17.else if $p$ is a SYN/ACK or RST/ACK, then
18.    $c \leftarrow$ FT_search($ICT$, $p.FID$)
19.else $c \leftarrow$ FT_search($ECT$, $p.FID$)
29.if $c = NULL$ and $p$ is a pure ACK, then
21.    $c \leftarrow$ FT_search($ICT$, $p.FID$)
22.if $c \neq NULL$, then
23.    Forward the packet $p$ to the network processor in the connection $c$, and update the connection $c$
with the packet $p$
24.    if $c$ is not yet established, then
25.        ICT_CBF_insert($c$)
26.    else if $c$ is terminated, then
27.        delete $c$
28.    else FT_insert($ECT$, $c$)
29.    return 1
30.else return 0
```

Table 4: The flow table timeout of the DTP-ICM scheme

Pseudo-code for the timeout scanning of a flow table

```
Alogrithm 2 Timeout(FlowTable table)
1. if table is the ICT, then
2.     if $table.size > ICT\_MAX\_SIZE$, then
3.         ICT_CBF_timeout(1);
4.     else
5.         ICT_CBF_timeout(4);
6. else if table is the ECT, then
7.     FT_timeout(ECT, 60);
8. else
9.     FT_timeout(table, 60);
```

and the rear inactive flow zone.

510 Suppose a flow table has the load factor α, and β is the ratio of the amount of active flows to its table length. With simple uniform distribution assumption, each list in the table can be deemed to consist of β flows in the front active zone and $\alpha - \beta$ ones in the rear inactive zone. As for a flow with n trains the ith of which contains m_i packets, we can count the search length for each train. As for the ith train, when its first packet comes, its flow is staying in the rear inactive zone. Thus we need take β to travel through the front active zone and $(\alpha - \beta + 1)/2$ to find out the flow in the rear inactive zone. Therefore, the first packet of the ith train needs to take the search length $SL_{i,0}(0 \leq i \leq n-1)$ in (5) to locate its flow.

$$SL_{i,0} = \beta + \frac{\alpha - \beta + 1}{2} = \frac{\alpha + \beta + 1}{2}. \tag{5}$$

As for each subsequent packet within the train, the flow has been shifted to the front active zone at the arrival of its first packet. Hence we can directly find out the flow in the front active zone with the search length $SL_{i,j}(0 \leq i \leq n-1, 1 \leq j \leq m_i - 1)$ in (6).

$$SL_{i,j} = \frac{\beta + 1}{2}. \tag{6}$$

In summary, we can get the total search length for the ith train $TSL_i(0 \leq$

$i \leq n-1$) in (7).

$$TSL_i = \sum_{j=0}^{m_i-1} SL_{i,j} = SL_{i,0} + \sum_{j=1}^{m_i-1} SL_{i,j} = \frac{\alpha + m_i(\beta+1)}{2}. \tag{7}$$

Suppose the average size of all packet trains is denoted as $m = \frac{1}{n}\sum_{i=0}^{n-1} m_i$, we can compute the average search length ASL in (8).

$$ASL = \frac{\sum_{i=0}^{n-1} TSL_i}{\sum_{i=0}^{n-1} m_i} = \frac{\beta+1}{2} + \frac{\alpha}{2m}. \tag{8}$$

As seen from (8), the lookup efficiency of a flow table chiefly depends on its load factor α, the average number of active flows in each list β, and the average size of all packet trains m. The length of the flow table is generally configured as 2^{16}. There are usually no more than 2^{20} simultaneous flows in high-speed networks, and α is expected to be less than 2^4. The amount of active flows is measured as no more than 2^{16}, and β is reckoned to value less than 1. The packet train size varies widely from several to more than a hundred, and m is supposed to be larger than 2^3. In summary, we can conclude that the ASL is likely to be smaller than 2. In contrast, a naive flow table performs lookup with the average search length $(\alpha+1)/2$, which is much larger than that of our proposed scheme in (8).

6. Experiments

This section evaluates the performance of our proposed traffic partitioning scheme DTP-ICM with physical network traffic traces in terms of traffic load balance and packet distribution performance.

6.1. Traffic Trace Properties

For convenient multiple evaluation and comparison, we operate network traffic partitioning offline on traffic traces previously captured from high-speed network links. The traces for our evaluation should contain a large number of simultaneous flows. Unfortunately, most published traffic traces are unsatisfactory because of anonymization. Eventually we employ the traffic traces[37] collected from a 10Gps main channel in the CERNET (Jiangsu). In particular, we select 4 traffic traces, each with 15,420,235 packets, whose properties are illustrated in *Table.5*.

Table 5: The properties of the 4 traffic traces used in our experiments.

Traffic trace	Duration	TCP packets	UDP packets	Initial connections	Established connections	Terminated connections	UDP sessions
TRACE20141109	35.4s	8.84M	6.49M	307K	75.7K	61.9K	266K
TRACE20140509	58.3s	9.21M	6.09M	195K	92.6K	74.7K	125K
TRACE20120921	86.4s	11.5M	3.52M	324K	142K	139K	72.9K
TRACE20120725	83.7s	9.79M	5.05M	318K	95.7K	89.9K	77.3K

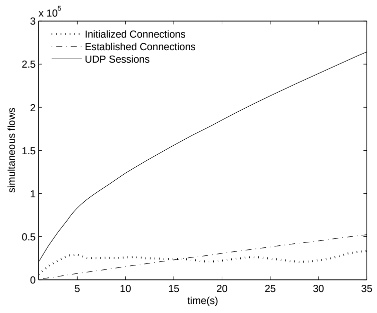
Fig.5 shows the number of simultaneous flows in the traces, which has a significant impact on the performance of dynamic traffic partitioning scheme. The number is counted by setting a timeout interval for each type of flow. In particular, we respectively set 4s, 60s and 60s as the timeout interval of initial TCP connections, established TCP connections and UDP sessions. As seen from *Fig.5*, the number of initial connections usually falls between 10K and 35K, and that of established connections grows from 15K to 50K in a slow rate. In stark contrast, the number of UDP sessions sharply increases from 60K to more than 260K. This is probably because of the popularity of UDP-based network applications such as VOIP, IPTV and online video websites.

6.2. Traffic Load Balance

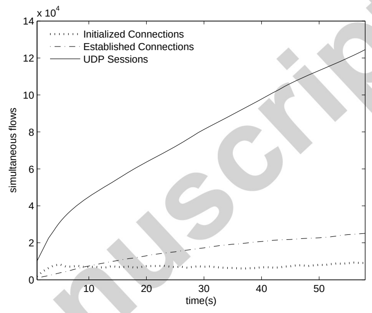
We contrast among load balancing capacities of typical traffic partitioning schemes based on the definition of load imbalance degree.

1) Load Imbalance Degree

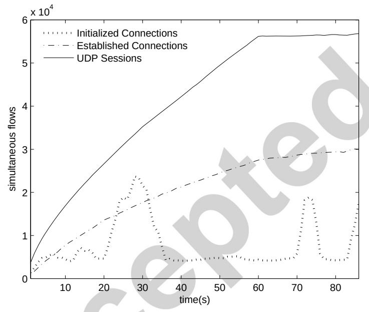
To quantize the load balance extent of network traffic partitioning, we define load imbalance degree by employing sample variance in mathematical statistics[24]. We first define the ideal probability $p_i (0 \le i \le N-1)$ of a packet distributed to the ith network processor as the benchmark of the balance evaluation. In particular, p_i can be calculated as the weight of the capacity C_i of the ith network processor among total capacity of all network processors in (9).



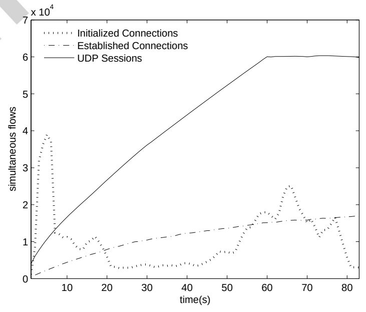
(a) TRACE20141109



(b) TRACE20140509



(c) TRACE20120921



(d) TRACE20120725

Figure 5: The number of simultaneous flows in the traffic traces.

$$p_i = \frac{C_i}{\sum_{j=0}^{N-1} C_j}. \tag{9}$$

Similarly, we further define the actual probability $p_i'(t)$ of a packet distributed to the ith network processor at time t. In particular, $p_i'(t)$ can be computed as the weight of the load $L_i(t)$ of the ith network processor among total load of all network processors at time t in (10).

$$p_i'(t) = \frac{L_i(t)}{\sum_{j=0}^{N-1} L_j(t)}. \tag{10}$$

With the above definitions, we define the following load imbalance degree $I(t)$ of traffic partitioning at time t in virtue of the concept and formula of sample standard deviation.

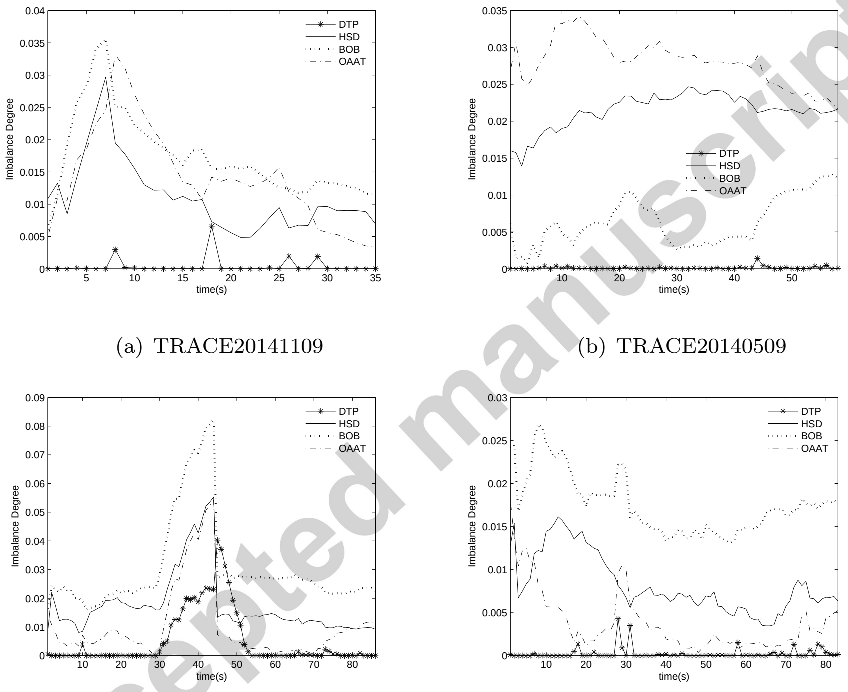
$$I(t) = \sqrt{\frac{1}{N-1}\sum_{i=0}^{N-1}\left(p_i'(t) - p_i\right)^2}. \tag{11}$$

As seen from the above definition, $I(t)$ characterizes the real-time load balance extent of network traffic partitioning ($0 \le I(t) \le 1$). The more $I(t)$ comes close to 0, the more the traffic load distributes evenly.

2) Load Balance Comparison

We compare the load balance capacity of dynamic traffic partitioning scheme (DTP) with those of the traditional ones, i.e., direct hashing and hash space division (HSD). The key to the direct hashing schemes is to choose a uniform hash function with low cost. Network investigations indicate that BOB and OAAT are the best candidates for packet processing among a collection of 25 hash functions[39][40]. So BOB and OAAT are selected as the representatives of the direct hashing schemes. Besides, we select the MD5 algorithm for the hash function in the HSD scheme.

Suppose there are 3 network processors, whose processing capacities are configured with the ratio 1:1:1 and 1:2:1 respectively. The traffic load of a network processor is simply modeled as the addition of the quadruple number of simultaneous flows and the amount of distributed packets. Our experiments are performed by reading packets from each traffic trace in their arrival orders and distributing them in terms of different traffic partitioning schemes. Then, we record the number of packets and flows distributed to each network processor, and calculate the load imbalance degree for each traffic partitioning scheme according to (11). Finally, we get the load imbalance degrees of these traffic partitioning schemes in *Fig.6* and *Fig.7*.

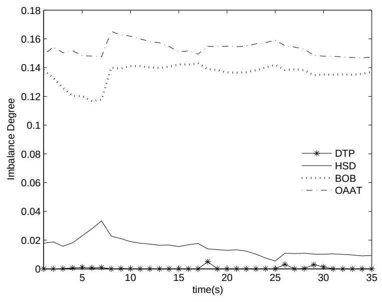


(a) TRACE20141109 (b) TRACE20140509

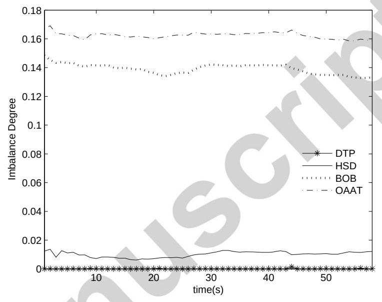
(c) TRACE20120921 (d) TRACE20120725

Figure 6: The load imbalance degree in the case of 3 network processors configured with the capacity ratio 1:1:1.

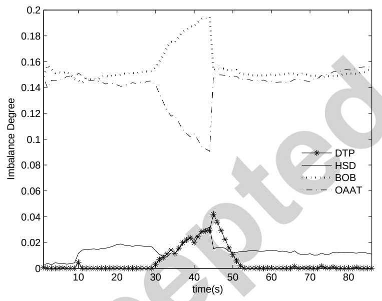
As seen from *Fig.6* and *Fig.7*, the DTP scheme has better load balance capacity than the other ones whether network processors are configured with identical capacity or not. In particular, the DTP scheme almost always has the lowest load imbalance degrees, and achieves ideal load balance effect in most cases. This is attributed to the fact that it dynamically distributes packets for



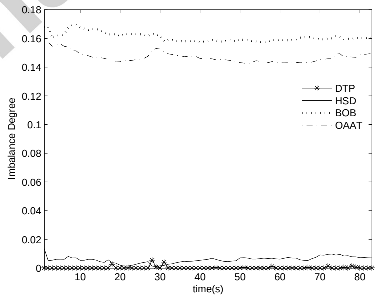
(a) TRACE20141109



(b) TRACE20140509



(c) TRACE20120921



(d) TRACE20120725

Figure 7: The load imbalance degree in the case of 3 network processors configured with the capacity ratio 1:2:1.

the purpose of the real-time balancing of traffic load distribution.

By contrast of *Fig.6* and *Fig.7*, we can see that both the DTP and the HSD schemes have excellent adaptability to different network processor configurations. In particular, their load imbalance degrees always fall below 0.25 whether the 3 network processors are configured with the capacity ratio 1:1:1 or 1:2:1. This effect is derived from that they adaptively distribute packets in terms of network processor capacities. Owing to this property, both the schemes have much better load balance effect than the BOB and the OAAT for network processors with different capacities, while the case is completely different for same network processors.

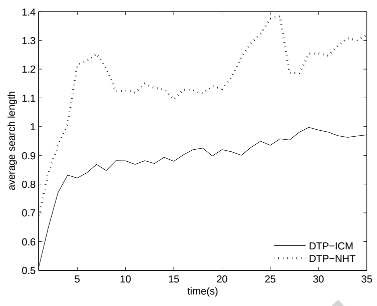
6.3. Packet Distribution Performance

Dynamic traffic partitioning scheme achieves excellent load balance effect at the cost of heavy packet distribution overheads. So it is necessary to compare our proposed dynamic traffic partitioning scheme DTP-ICM with the traditional one in terms of packet distribution performance. The traditional scheme[22][23][21][24] maintains simultaneous flows of each protocol above IP header in a naive hash table, called DTP-NHT. As for flow tables in both schemes, hash functions are all simply devised as the XOR operation of the flow identifier folded by 16 bits for low cost, and their table lengths are uniformly sets as the maximum 2^{16}. Besides, the counting bloom filter in our DTP-ICM scheme employs 6 hash functions, BOB, OAAT, TWMX, RS, Hsieh, and SBox[40].

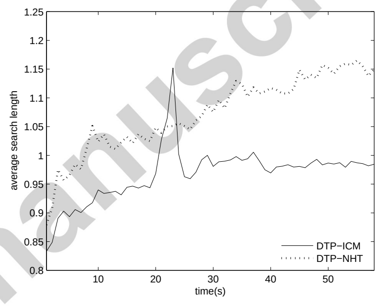
As seen from the pseudo-code implementation in the previous section, the flow table lookups dominate the packet distribution performance of dynamic traffic partitioning scheme. Thus, the average search length (ASL) is employed to characterize the packet distribution performance, due to its independence of testing platforms. In particular, we count the search length and the number of incoming packets, and calculate their quotient as the average search length.

Experiment 1 (*Normal Conditions*)

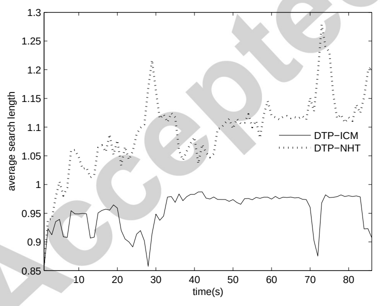
We first compare the packet distribution performance of both dynamic traffic partitioning schemes with the selected traces under normal conditions. The counting bloom filter in our DTP-ICM scheme is configured in terms of its equation $m = nklog_2e$ in the Section 4.3, where m, k and n respectively denotes the number of counters, hash functions and initial connections in the ICT table. As seen from *Fig.5*, the number of initial connections n is no more than 30K in most cases. Then, the number of counters m is suitable to be set as 256K since the number of hash functions k equals to 6. With the above configurations, we operate both dynamic traffic partitioning schemes on the selected traffic traces, and evaluate the ASLs of both schemes for TCP and UDP traffic in *Fig.8* and *Fig.9* respectively.



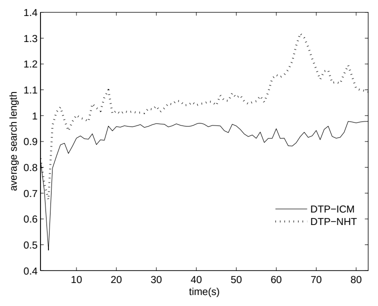
(a) TRACE20141109



(b) TRACE20140509



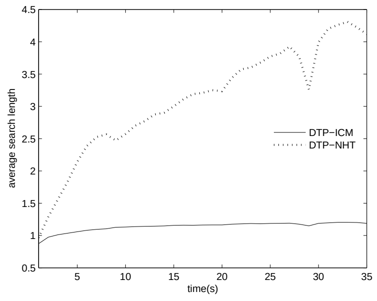
(c) TRACE20120921



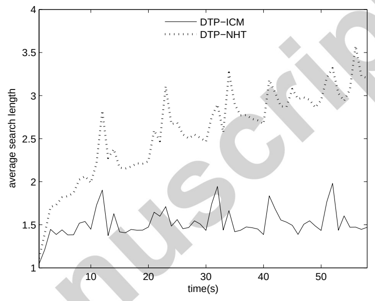
(d) TRACE20120725

Figure 8: The ASLs of TCP traffic partitioning.

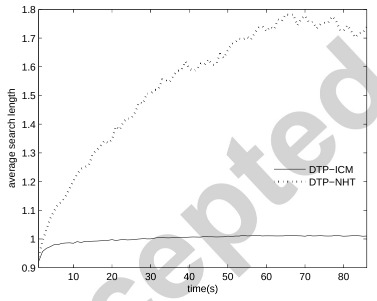
As seen from *Fig.8* and *Fig.9*, our proposed DTP-ICM scheme outperforms the DTP-NHT scheme for both TCP and UDP traffic. As for TCP traffic, our DTP-ICM scheme performs packet distribution with the ASLs below 1 almost



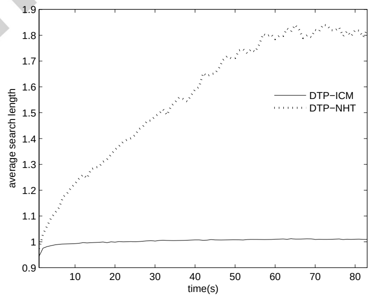
(a) TRACE20141109



(b) TRACE20140509



(c) TRACE20120921



(d) TRACE20120725

Figure 9: The ASLs of UDP traffic partitioning.

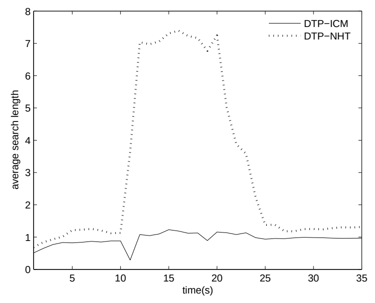
at all times, while the ASLs of the DTP-NHT scheme always fluctuate above 1. This is chiefly attributed to our optimizations of TCP connection management, i.e., the initial connection separation mechanism and the move-to-front heuristic. In particular, most TCP packets only need to look up one of the ICT and the ECT tables divided from the original TCP connection table, and the successful search length of a TCP packet is greatly shorten for the ECT table as a result of network traffic locality. As for UDP traffic, the ASLs of the DTP-NHT scheme rise up sharply due to the increasing number of simultaneous UDP flows. As a striking contrast, our DTP-ICM scheme perform with much more steady and shorter ASLs. This result thanks to the move-to-front heuristic applied to the UDP session table.

Experiment 2 (*SYN Flooding Attacks*)

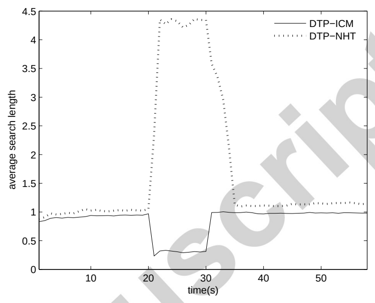
SYN flooding attacks often have a great destructive power on dynamic traffic partitioning scheme. We compare the packet distribution performance of both dynamic traffic partitioning schemes under the attacks. The attacks are simulated by inserting falsified SYN packets into normal TCP traffic in our selected traces. In particular, we mix normal packets in the trace with falsified packets by different hybrid ratio to simulate different levels of the attacks.

Our counting bloom filter is still configured in terms of the equation $m = nklog_2e$ under SYN flooding attacks. However, its parameter configuration varies with different attack levels due to its impact on the number of initial connections in the ICT table n. The table during the attacks is dominated by unexpired false connections, whose amount depends on the timeout interval of each initial connection, the number of normal packets per second, and the hybrid ratio of falsified packets and normal packets. The timeout interval is generally adjusted to 1 second in the presence of the attacks[26][27]. The number of normal packets per second in the trace is around 256K at all times illustrated in *Table.5*. Besides, the number of hash functions k is fixed as 6. In summary, the number of counters m can be configured as 2M times of the packet hybrid ratio.

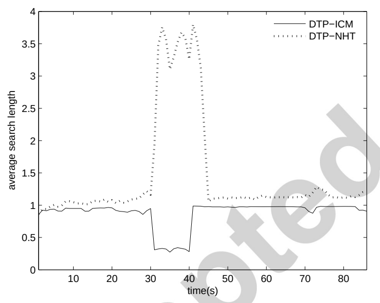
In our experiments, the falsified SYN packets are inserted into packet stream in the TRACE20141109, TRACE20140509, TRACE20120921 and TRACE20120725 respectively from the 10th, 20th, 30th and 30th second. Then we operate both dynamic traffic partitioning schemes on the hybrid packet traffic. *Fig.10*, *Fig.11* and *Fig.12* demonstrate the ASLs of both schemes with the packet hybrid ratio 1:1, 2:1 and 4:1 respectively.



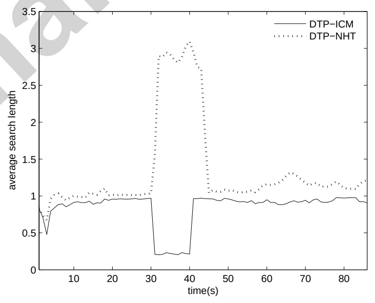
(a) TRACE20141109



(b) TRACE20140509



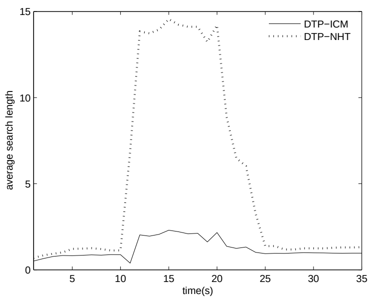
(c) TRACE20120921



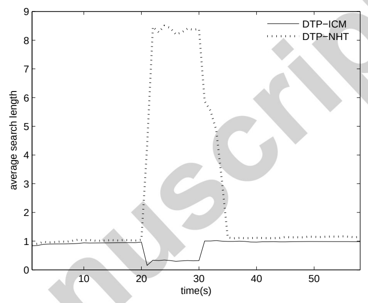
(d) TRACE20120725

Figure 10: The ASLs with the packet hybrid ratio 1:1.

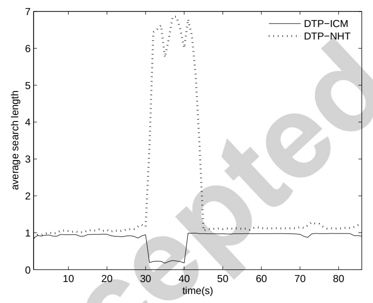
As seen from *Fig.10*, *Fig.11* and *Fig.12*, our proposed DTP-ICM scheme performs packet distribution at much shorter ASLs than the traditional DTP-NHT scheme. In particular, the DTP-ICM scheme always takes steady and short search lengths no matter how fierce the attacks are. In contrast, the DTP-NHT scheme degrades sharply in the emergence of SYN flooding attacks. What is more exciting is that, the DTP-ICM scheme get even shorter ASLs when the attacks become more severe. In summary, our proposed DTP-ICM



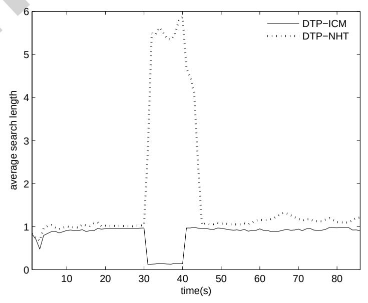
(a) TRACE20141109



(b) TRACE20140509

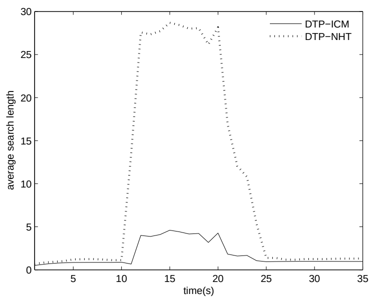


(c) TRACE20120921

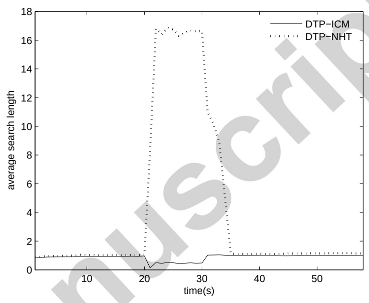


(d) TRACE20120725

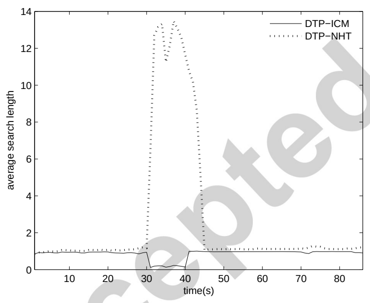
Figure 11: The ASLs with the packet hybrid ratio 2:1.



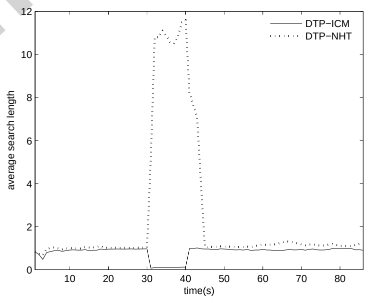
(a) TRACE20141109



(b) TRACE20140509



(c) TRACE20120921



(d) TRACE20120725

Figure 12: The ASLs with the packet hybrid ratio 4:1.

scheme provides much better robustness against SYN flooding attacks than the traditional one. This is attributed to the fact that all falsified SYN packets no longer need to search any flow tables due to the application of the counting bloom filter to the ICT table.

7. Conclusion and Future Work

This paper proposes a novel dynamic traffic partitioning scheme DTP-ICM, which assigns a new flow to a network processor with the lightest traffic load level and distributes all packets within a flow to its network processor, to guarantee flow granularity and achieve good load balance. In the scheme, we build the initial connection separation mechanism to isolate false TCP connections induced by malicious behaviors, which results in two connection tables, the ICT and the ECT. The move-to-front heuristic is applied to the ECT table and the UDP flow table to accelerate their lookups and to the ICT table to speed up its timeout scanning. More importantly, the ICT table is optimized with counting bloom filters to defend against SYN flooding attacks.

Our proposed DTP-ICM scheme is evaluated with real network traffic traces. The evaluation results indicate that the DTP-ICM scheme achieves good load balance effect and high packet distribution performance. In particular, the DTP-ICM scheme perform packet distribution with steady and short ASLs below 1 almost at all times for both TCP and UDP traffic. More excitingly, the DTP-ICM scheme still takes steady and short search lengths no matter how fierce the attacks are.

In our future work, more traffic traces from different high-speed network lines will be utilized to evaluate and validate our proposed traffic partitioning scheme. After that, we plan to deploy it in specific network applications such as content delivery. Furthermore, applications of our proposed scheme to the environment of future networks, such as software defined networks, is also within our future work plan.

Acknowledgment

This work was supported in part by National Natural Science Foundation of China (61502056, 61303043), Hunan Provincial Natural Science Foundation of China (2015JJ3010, 13JJ4052), and Scientific Research Fund of Hunan Provincial Education Department (15B009, 14C0285).

References

[1] E.W. Fulp, R.J. Farley. A Function-Parallel Architecture for High-Speed Firewalls. In: Proceedings of IEEE International Conference on Communications, Istanbul, 2006. 2213-2218.

[2] A. Patel, M. Taghavi, K. Bakhtiyari, J. C. Junior. An intrusion detection and prevention system in cloud computing: A systematic review. Journal of Network and Computer Applications, 2013, 36(1): 25-41.

[3] G. Vasiliadis, M. Polychronakis, S. Ioannidis. MIDeA: a multi-parallel intrusion detection architecture. In: Proceedings of ACM conference on Computer and Communications Security. 2011. 297-308.

[4] G. Cheng, J. Gong, W. Ding. Distributed sampling measurement model in a high speed network based on statistical analysis. Chinese Journal of Computers, 2003, 26(10): 1266-1273.

[5] Z. H. Zhang. Distribute and Match - The DM Switch for High Speed Packet Switching Networks. In: Proceedings of IEEE Global Telecommunications Conference, 2011. 1-6.

[6] S. Manfredi, F. Oliviero, S. P. Romano. A Distributed Control Law for Load Balancing in Content Delivery Networks. IEEE/ACM Transactions on Networking, 2012, 21(1): 55-68.

[7] N. Mohamed, J. Al-Jaroodi, A. Eid. A dual-direction technique for fast file downloads with dynamic load balancing in the Cloud. Journal of Network and Computer Applications, 2013, 36(4): 1116-1130.

[8] A. Hakiri, A. Gokhale, P. Berthou, D. C. Schmidt, T. Gayraud. Software-Defined Networking: Challenges and research opportunities for Future Internet. Computer Networks, 2014, 75(24): 453-471.

[9] D. Kreutz, F. M. V. Ramos, P. Verissimo, E. C. Rothenberg, S. Azodolmolky, S. Uhlig. Software-defined networking: A comprehensive survey. Proceedings of the IEEE, 2015, 103(1): 14-76.

[10] M. Koerner, O. Kao. Multiple Service Load-Balancing with OpenFlow. In: Proceedings of the IEEE 13th International Conference on High Performance Switching and Routing (HPSR), 2012: 210-214.

[11] M. Bredel, Z. Bozakovy, A. Barczyk, H. Newman. Flow-Based Load Balancing in Multipathed Layer-2 Networks using OpenFlow and Multipath-TCP. In: Proceedings of the 3rd ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN), 2014: 213-214.

[12] Y. Qi, B. Xu, F. He, B. Yang, J. M. Yu, J. Li. Towards high-performance flow-level packet processing on multi-core network processors. In: Proceedings of ACM/IEEE Symposium on Architectures for Networking and Communications Systems. Orlando, 2007. 17-26.

[13] Z. H. Zhao, Y. T. Shu, L. F. Zhang, H. M. Wang, O. W. W. Yang. Flow-level multipath load balancing in MPLS network. In: Proceedings of IEEE International Conference on Communications, Paris, 2004, 2: 1222-1226.

[14] A. Kirsch, M. Mitzenmacher, G. Varghese. Hash-based techniques for high-speed packet processing. Algorithms for Next Generation Networks (Computer Communications and Networks), 2010, 2:181-218.

[15] Z. Cao, Z. Wang, E. Zegura. Performance of Hashing-based Schemes for Internet Load Balancing. In: Proceedings of IEEE International Conference on Computer Communications. Tel-Aviv, Israel, 2000: 332-341.

[16] H. Lai, H. Huang, J. Xie. PABCS: A Traffic Partition Algorithm for Parallel Intrusion Detection. Chinese Journal of Computers, 2007, 30(4): 555-562.

[17] Y. Chen, X. Lu, X. Shi, et al. A Session-oriented Adaptive Load Balancing Algorithm. Journal of Software, 2008, 19(7): 1828-1836.

[18] Q. Sun, D. Zhang, P. Gao, et al. Study of Parallel IDS Load Balancing Algorithm. Mini-micro Systems, 2004, 25(12): 2215-2217.

[19] W. Shi, M. H. MacGregor, P. Gburzynski. Load Balancing for Parallel Forwarding. IEEE/ACM Transactions on Networking, 2005, 13(4): 790-801.

[20] L. Kencl, J. L. Boudec. Adaptive Load Sharing for Network Processors. IEEE/ACM Transactions on Networking, 2008, 16(2): 293-306.

[21] X. Li, D. Zhao, H. Zhao, et al. Research on Application-based Network Intrusion Detection System for High-speed Network. Journal of China Institute of Communications, 2002, 23(9): 1-7.

[22] W. Jiang, H. Song, Y. Dai. Real-time Intrusion Detection for High-speed Networks. Computers and Security, 2005, 24(4): 287 294.

[23] W. Jiang, S. Hao, Y. Dai, et al. Load Balancing Algorithm for High-speed Network Intrusion Detection Systems. Journal of Tsinghua University (Science and Technology), 2006, 46(1): 106-110.

[24] B. Xiong, H. Xiao, M. Long, et al. Dynamic Partitioning of High-speed Network Traffic with Flow Level. Mini-micro Systems, 2013, 34(5): 945-950.

[25] K. Thompson, G. J. Miller, R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics. IEEE Network, 1997, 11: 10-23.

[26] H. Kim, J. Kim, I. Kang, et al. Preventing Session Table Explosion in Packet Inspection Computers. IEEE Transactions on Computers, 2005, 54(2): 238-240.

[27] I. Kang, H. Kim. Determining Embryonic Connection Timeout in Stateful Inspection. In: Proceedings of IEEE International Conference on Communications, Anchorage, Alaska, 2003: 458-462.

[28] C. Williamson. Internet Traffic Measurement. IEEE Internet Computing, 2001, 5(6): 70-74.

[29] N. Gulati, C. Williamson, R. Bunt. LAN Traffic Locality: Characterization and Application. In: Proceedings of International Conference in Local Area Network Interconnection. North Carolina, 1993: 233-250.

[30] Y. Liu, L. Guo, F. Li, et al. A Case Study of Traffic Locality in Internet P2P Live Streaming Systems. In: Proceedings of International Conference on Distributed Computing Systems. Montreal, Canada, 2009: 423-432.

[31] H. Wang, J. Liu, B. Chen, et al. On Tracker Selection for Peer-to-Peer Traffic Locality. In: Proceedings of International Conference on Peer-to-Peer Computing. Delft, Netherlands, 2010: 1-10.

[32] B. Xiong, F. Li, L. Jiang, et al. An efficient oriented-hash table for connection management in high-speed networks. Journal of Huazhong University of Science and Technology (Natural Science Edition), 2011, 39(2): 19-22.

[33] B. Bloom, Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970, 13(7): 422-426.

[34] L. Fan, P. Cao, J. Almeida, et al. Summary cache: A scalable wide-area web cache sharing protocol. IEEE/ACM Transactions on Networking, 2000, 8(3):281-293.

[35] S. Cohen, Y. Matias. Spectral Bloom Filters. In: Proceedings of 2003 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2003. 241-252.

[36] K. R. Fall, R. W. Stevens. TCP/IP Illustrated, Volume 1: The Protocols (2nd Edition). Boston, Massachusetts: Addison-Wesley, 2011.

[37] Network traffic traces. `http://iptas.edu.cn/src/system.php`. 2015.

[38] B. Xiong, K. Yang, F. Li, et al. The Impact of Bitwise Operators on Hash Uniformity in Network Packet Processing. International Journal of Communication Systems, 2014, 27(11): 3158-3184.

[39] C. Henke, C. Schmoll, T. Zseby. Empirical evaluation of hash functions for multipoint measurements. ACM SIGCOMM Computer Communication Review, 2008, 38(3): 41-50.

[40] M. Molina, S. Niccolini, N.G. Duffield. A comparative experimental study of hash functions applied to packet sampling. In: Proceedings of International Teletraffic Congress (ITC-19), Beijing, 2005. 1-11.