Journal of Network and Computer Applications 87 (2017) 20–31

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Robust dynamic network traffic partitioning against malicious attacks

Bing Xiong[a,b,*], 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*

ARTICLE INFO

Article history:
Received 23 November 2014
Received in revised form
30 October 2015
Accepted 16 April 2016
Available online 7 June 2016

Keywords:
Dynamic traffic partitioning
SYN flooding attacks
Half-open connection separation
Counting bloom filters

2015 MSC:
00-01
99-00

ABSTRACT

The continual growth of network traffic rates leads to heavy packet processing overheads, and a typical solution is to partition traffic into multiple network processors for parallel processing especially in emerging software-defined networks. This paper is thus motivated to propose a robust dynamic network traffic partitioning scheme to defend against malicious attacks. After introducing the conceptual framework of dynamic network traffic partitioning based on flow tables, we strengthen its TCP connection management by building a half-open connection separation mechanism to isolate false connections in the initial connection table (ICT). Then, the lookup performance of the ICT table is reinforced by applying counting bloom filters to cope with malicious behaviors such as SYN flooding attacks. Finally, we evaluate the performance of our proposed traffic partitioning scheme with real network traffic traces and simulated malicious traffic by experiments. Experimental results indicate that our proposed scheme outperforms the conventional ones in terms of packet distribution performance especially robustness against malicious attacks.

1. Introduction

The continual growth of network bandwidth results in the fact that a single network device cannot real-timely process large-scale packet traffic, especially in high-speed networks with bandwidth 1 Gps 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 (Fulp and Farley, 2006), intrusion detection (Patel et al., 2013; Vasiliadis et al., 2011), traffic measurement (Cheng et al., 2003), high-speed packet switching (Zhang, 2011), and content delivery (Manfredi et al., 2012; Mohamed et al., 2013). In particular, emerging network paradigms such as software-defined networking (Hakiri et al., 2014; Kreutz et al., 2015) make load balancing much easier to be deployed without changing substrate devices by using OpenFlow (Koerner and Kao, 2012; Bredel et al., 2014).

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 (Qi et al., 2007; Zhao et al., 2004). In particular, all packets within a flow must be assigned to an identical network processor to guarantee session integrity. For convenience, the terms flow and session in this paper both refer to packet traffic between two network endpoints, especially called connection for TCP. Secondly, network traffic ought to be dynamically partitioned in terms of the processing capacities of network processors to achieve good load balance. Thirdly, each packet should be distributed at fast speeds to real-timely process large-scale network traffic even in the presence of malicious attacks.

Existing network traffic partitioning schemes can be classified into three types: direct hashing (Kirsch et al., 2010; Jo and Kim, 2004; Cao et al., 2000), hash space division (Sun et al., 2004) and dynamic schemes based on flow tables (Li et al., 2002; Jiang et al., 2005, 2006; Xiong et al., 2013). 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 dynamic schemes maintain a hash table of simultaneous flows,

* Corresponding author at: School of Computer and Communication Engineering, Changsha University of Science and Technology, Changsha 410114, China.
E-mail addresses: xiongbing@csust.edu.cn, xiongbing.csust@qq.com (B. Xiong).

http://dx.doi.org/10.1016/j.jnca.2016.04.013
1084-8045/© 2017 Elsevier Ltd. All rights reserved.

and dynamically distribute packet traffic with flow granularity to multiple network processors. This type keeps flow integrity during packet distribution and adequately balances traffic load among network processors. However, their packet distribution performance becomes a great challenge especially when adversaries launch malicious behaviors such as SYN flooding attacks.

Many literatures have achieved deep insights into the load balancing capacity of dynamic network traffic partitioning. Chen et al. (2008) 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. (2004) mapped each incoming packet by hashing its key header fields to an interval allocated for a 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. (2005) classified Internet flows into two categories: the aggressive and the normal, and applied dynamic scheduling policies to the aggressive flows. Kencl and Boudec (2008) 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 pay little attention to packet distribution performance especially robustness against malicious behaviors.

This paper focuses on how to promote the packet distribution performance of dynamic network traffic partitioning. In particular, the promotion is investigated with the following methodology. We first give the conceptual framework of dynamic traffic partitioning scheme based on flow tables. Then, its TCP connection management is strengthened by isolating false connections to make them much easier to control. As a further step, the management of false connections is reinforced to mitigate the destructive effects of malicious attacks especially on the lookup performance of TCP connection tables. Next, we give the algorithmic implementation of our dynamic traffic partitioning scheme based on packet classification in terms of TCP connection tables. Finally, the packet distribution performance of our scheme is evaluated by carrying out experiments with real network traffic traces and simulated malicious traffic.

With the above methodology, we aim to achieve the following conclusions as the main contributions of this paper: (a) giving the conceptual framework of dynamic traffic partitioning scheme based on flow tables to guarantee flow granularity and achieve real-time traffic load balance; (b) strengthening TCP connection management by building a half-connection connection separation mechanism to isolate false TCP connections in the initial connection table (ICT); (c) reinforcing the lookup performance of the ICT table by applying counting bloom filters to defend against malicious behaviors such as SYN flooding attacks.

The rest of this paper is organized as follows. Section 2 introduces the related work. In Section 3, we describe the conceptual framework of dynamic network traffic partitioning based flow tables, and promote its packet distribution performance by building the half-open connection separation and employing counting bloom filters to resist malicious attacks. Section 4 gives the algorithmic implementation and complexity analysis of our proposed dynamic network traffic partitioning scheme. In Section 5, we evaluate the packet distribution performance of our proposed scheme with real network traffic traces. Section 6 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. (2000) 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 (Shi et al., 2005). Lai et al. (2007) 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. (2008) 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. (2004) 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. (2005) 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 and Boudec (2008) 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. (2002) 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. (2005, 2006) 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. (2013) dynamically maintained a hash table of concurrent sessions, and assigned a session to a network processor with the lightest load level when the session appeared. These algorithms achieve good effect of traffic load balancing, but their packet distribution performance is not adequately considered.

For the above motivations, this paper proposes a robust dynamic network traffic partitioning scheme. In this scheme, we strengthen its TCP connection management by building the half-open connection separation mechanism to isolate false TCP connections in the ICT table and reinforce the lookup performance of the ICT table by employing counting bloom filters to defend against SYN flooding attacks. By this way, we strive to boost the packet distribution performance of dynamic traffic partitioning scheme especially in the presence of malicious attacks.

3. Dynamic traffic partitioning scheme

This section describes the conceptual framework of dynamic traffic partitioning based on flow tables, and optimizes its TCP connection management by building the half-open connection separation mechanism and employing counting bloom filters for robustness against malicious attacks.

3.1. Conceptual framework

Many network applications relating to stateful packet processing usually employ load balancing technologies 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 (Qi et al., 2007): (a) flow granularity. Network traffic must be partitioned with flow granularity to support packet processing at semantic levels; (b) dynamic load balance. Network traffic ought to be dynamically partitioned to achieve real-time load balance among all network processors; (c) good packet distribution performance. Each arrived packet should be distributed at fast speeds to real-timely process large-scale network traffic even in the presence of malicious attacks.

The first two properties can be adequately satisfied by dynamic network traffic partitioning schemes based on flow tables, whose basic idea is explained as follows. These schemes 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, a new flow is assigned 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, the load state table is built to real-timely maintain the traffic load states of all network processors.

According to the above basic idea, we illustrate the fundamental principle of dynamic traffic partitioning scheme in Fig. 1. As for a continuous stream of arriving packets $(p_1, p_2, \ldots, p_i, \ldots)$, the traffic partitioner looks up in its flow table *FT* with the length *L* and forwards them one by one to *N* network processors in terms of the load state table *LST*. With regard to a packet p_i, we get its flow identifier from its header fields and map the identifier to a hash bucket of the flow table $FT_l(0 \le i \le L-1)$. Then, we look up the flow list in the bucket FT_l for a match $f_{l,k}$. If the lookup succeeds, we directly forward the packet to the network processor $NP_n(0 \le n \le N-1)$, whose number n is kept in the flow $f_{l,k}$. 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 $NP_j(0 \le j \le N-1)$ 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 *j* in the flow table *FT*. Finally, the packet p_i is forwarded to the network processor NP_j.

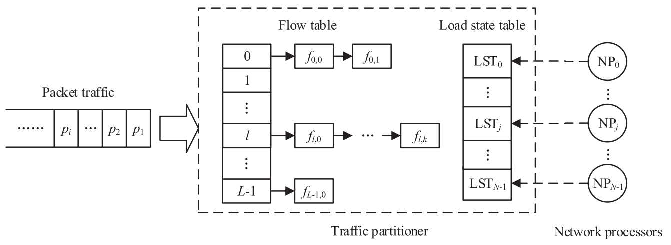


Fig. 1. The fundamental principle of dynamic traffic partitioning scheme.

Dynamic traffic partitioning at the micro level is to distribute each incoming packet to a network processor. As seen from the above fundamental principle, an essential operation of packet distribution is the flow table lookup, whose performance chiefly relies on flow table management. A conventional scheme for flow table management is to maintain all simultaneous flows of each protocol in the IP header, typically TCP and UDP, in a single hash table. This scheme works well for classical networks with 100 Mbps or even below, where the number of concurrent flows is limited and the flow tables keep in moderate sizes. However, the flow tables will begreatly enlarged when it comes to high-speed networks with bandwidth 1 Gbps and above, where there will be up to hundreds of thousands of simultaneous flows (Cai et al., 2014; Wolf et al., 2007; Park et al., 2000). Fortunately, their lookup overheads can still be moderately controlled by selecting a biggish flow table length. Suppose there are up toa hundred thousand flows for a transport-layer protocol, its flow table will hold the load factor slightly larger than 1 if its table length is set as 2^{16}, and its lookup overheads will be apparently acceptable with uniform hashing.

Unfortunately, the conventional 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 be confronted with great challenges under malicious attacks. Therefore, it is necessary to optimize the flow table management in dynamic traffic partitioning schemes for better packet distribution performance.

3.2. Half-open connection separation

Malicious behaviors such as SYN flooding attacks have a destructive effect 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. This expands the table 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 a half-open connection separation (HCS) mechanism to isolate false connections from normal connections. The essential concept of the HCS 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 within our connection management in 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.

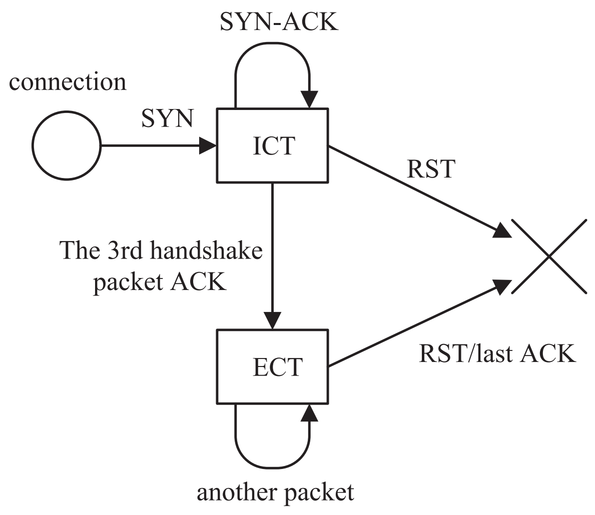


Fig. 2. The lifecycle of a connection in the HCS mechanism.

Subsequently, a TCP connection is manipulated as follows on the arrival of 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 HCS mechanism results in two controllable TCP connection tables, i.e., the ICT table and the ECT table. The ECT table is immune to malicious behavior in terms of the above working principle, and always keeps in an acceptable size usually no more than a hundred thousand according to Internet traffic measurements (Benetazzo et al., 2007; Williamson 2001; Thompson et al., 1997). The ICT table only 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 s and 4 s to complete three-way handshake (Kim et al., 2005; Kang and Kim, 2003). When malicious attacks appear, a continuous sequence of false connections will swarm into the ICT table, and we strive to eliminate them from theICT table as early as possible by setting a small timeout interval (e.g. 1 s). As a consequence, the ICT table maintains a limited number of false connections produced during the interval besides of normal initial connections. In summary, the HCS mechanism divides a oversize TCP connection table into two size-controllable tables, and makes malicious attacks much easier to resist.

3.3. Counting bloom filters

When malicious 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. Note that the lookups of the ICT table are destined to be failures for dominant SYN packets. Therefore, we can employ counting bloom filters to predict the lookup failures without searching the table.

A bloom filter (Bloom, 1970) is a simple space-efficient data structure for representing a set in order to support membership queries. A counting bloom filter (Fan et al., 2000) generalizes a bloom filter data structure so as to allow that the set can be changing dynamically via insertions and deletions. Fig. 3 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 $h_1, h_2, \ldots, h_k$, each with range $\{1, \ldots, m\}$. It has a set C of n elements $c_1, c_2, \ldots, c_n$, i.e., connection identifiers in the ICT table.

If an element c is to be inserted into the set C, the counters $A[h_i(c)](1 \le i \le k)$ at position $h_1(c), h_2(c), \ldots, h_k(c)$ 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 $h_1(c), h_2(c), \ldots, h_k(c)$ in A are decremented by 1 accordingly. If we want to query for an element c, we check all the value of the counters $A\left[h_i(c)\right](1 \le i \le k)$. 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 1 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 Fan et al. (2000) and Cohen and Matias (2003)

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

The right-hand expression in (1) is minimized for $k = \ln 2_* m/n$, in which 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

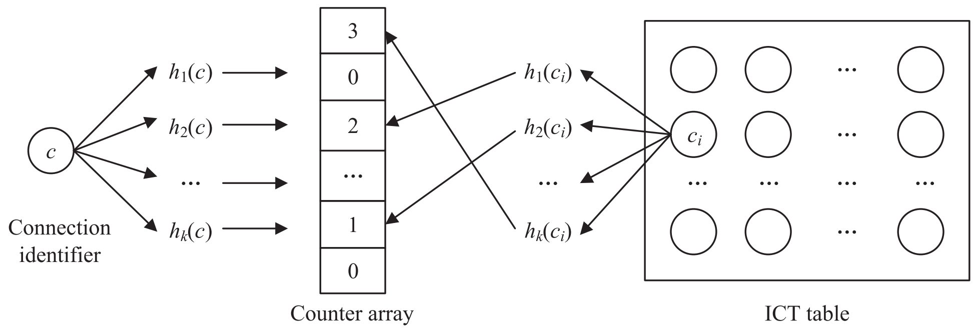


Fig. 3. The principle of the ICT table employing a counting bloom filter.

Table 1
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** $keyset \neq \varnothing$ **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		

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

As mentioned above, we can optimize the false positive rate with $k = \ln 2_{*}m/n$, so we assume that we restrict ourselves to $k = \ln 2_{*}m/n$, then

$$\Pr\left(\max_j A[j] \geq i\right) \leq m\left(\frac{e \ln 2}{i}\right)^i. \tag{3}$$

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

$$\Pr\left(\max_j A[j] \geq 16\right) \leq m\left(\frac{e \ln 2}{16}\right)^{16} \approx 1.37 \times 10^{-15} \times m. \tag{4}$$

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 foresee 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 decreased especially under SYN flooding attacks. By this way, we significantly mitigate the damage of malicious attacks on the lookup performance of flow tables.

4. Algorithmic implementation and analysis

This section describes the algorithmic implementation of our dynamic traffic partitioning scheme called DTP-CBF based on TCP packet classification, and analyzes its algorithmic complexity in terms of average search length.

4.1. 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 (Fall and Stevens, 2011): (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 aborts its connection in response to an unacceptable segment or a termination command from its application. These causes generally arise 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 acknowledgment of some transmitted segment during data transfer phase, whose connection is located in the ECT table; (c) an acknowledgment 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.

4.2. Algorithmic 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 4.1 (*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 4.2 (*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) $FEI_1{\cdot}IP < FEI_2{\cdot}IP$ or (b) $FEI_1{\cdot}IP = FEI_2{\cdot}IP$ and $FEI_1{\cdot}PT < FEI_2{\cdot}PT$.

Definition 4.3 (*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 Definition 4.2, 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 recorded 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 2 summarizes the above discussion regarding the packet distribution of our proposed DTP-CBF 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 s under normal conditions due to the fact that 99% of TCP connections take less than 4 s to complete three-way handshake. When malicious attacks occur, the interval is tuned to 1 s 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 s. Table 3 demonstrates the pseudo-code for the flow table timeout of our proposed DTP-CBF scheme.

Table 2
The packet distribution of the DTP-CBF scheme.

Pseudo-code for the distribution of a packet

```
Algorithm 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($ECT$, $p.FID$)
20. 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
```

packets are υ times that of normal packets; (d) the counting bloom filter determine the lookup result of a SYN packet with the false positive error rate *p*.

With the above assumptions, the ECT table is supposed to hold the load factor close to α, since the established connections dominate TCP connections under normal conditions. As seen form the algorithmic description, we only need to search the ECT table for all normal packets except three-way handshakes. Therefore, it is reckoned that a normal packet approximately takes the search length in (5) on average to find its connection.

$$ASL_{normal} = 1 + \alpha/2. \tag{5}$$

As for attack packets typically falsified SYN ones, we do not need to search any flow table if the counting bloom filter succeeds to predict their failed lookups. In such case, it is supposed to take

Table 3
The flow table timeout of the DTP-CBF 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);
```

4.3. Algorithmic complexity analysis

As shown in the above algorithmic description, flow table lookups dominate the packet distribution performance of our proposed dynamic traffic partitioning scheme DTP-CBF especially under malicious attacks. Therefore, we analyze the algorithm complexity of the DTF scheme for TCP traffic in terms of average search length. The analysis is carried out with the following assumptions: (a) TCP connections are uniformly distributed in a hash table with the load factor α under normal conditions; (b) a normal connection contain ω packets on average; (c) the number of attack

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

Traffic trace	Duration (s)	TCP packets (M)	Initial connections (K)	Established connections (K)	Terminated connections (K)	Total connections (M)
TRACE20110418	107	8.75	282	107	97.9	0.49
TRACE20140509	58.3	9.21	195	92.6	74.8	0.38
TRACE20120921	86.4	11.5	324	142	139	1.07
TRACE20120725	83.7	9.79	318	95.7	89.9	1.42

the search length 1 as the price of computing hashes for each packet. Otherwise, we should continue to search through the ICT table probably in failure. With the same hash table length of the ECT table, the ICT table is considered to keep the load factor $\upsilon\omega\alpha$. So an attack packet will take the search length $\upsilon\omega\alpha + 1$ for unsuccessful prediction of its lookup result. With the false positive error rate *p*, it can summarized that we need to take the average search length in (6) for attack packets.

$$ASL_{attack} = (1 - p)\cdot 1 + p(\upsilon\omega\alpha + 1) = p\upsilon\omega\alpha + 1. \tag{6}$$

In summary, we can calculate the average search length of our proposed scheme in (7) with (5) and (6).

$$\begin{aligned} ASL_{DTP-CBF} &= \frac{1}{\upsilon + 1} ASL_{normal} + \frac{\upsilon}{\upsilon + 1} ASL_{attack} \\ &= \frac{\left(p\upsilon^2\omega + 0.5\right)\alpha}{\upsilon + 1} + 1. \end{aligned} \tag{7}$$

As for conventional dynamic traffic partitioning schemes that maintain all simultaneous flows of each protocol above IP header in a single hash table, we need to match $(\alpha + \upsilon\omega\alpha)/2 + 1$ times on average in the TCP connection table for normal packets and take the average search length $\alpha + \upsilon\omega\alpha$ to go through the table for malicious packets. Consequently, we can compute the average search length of the conventional schemes in (8).

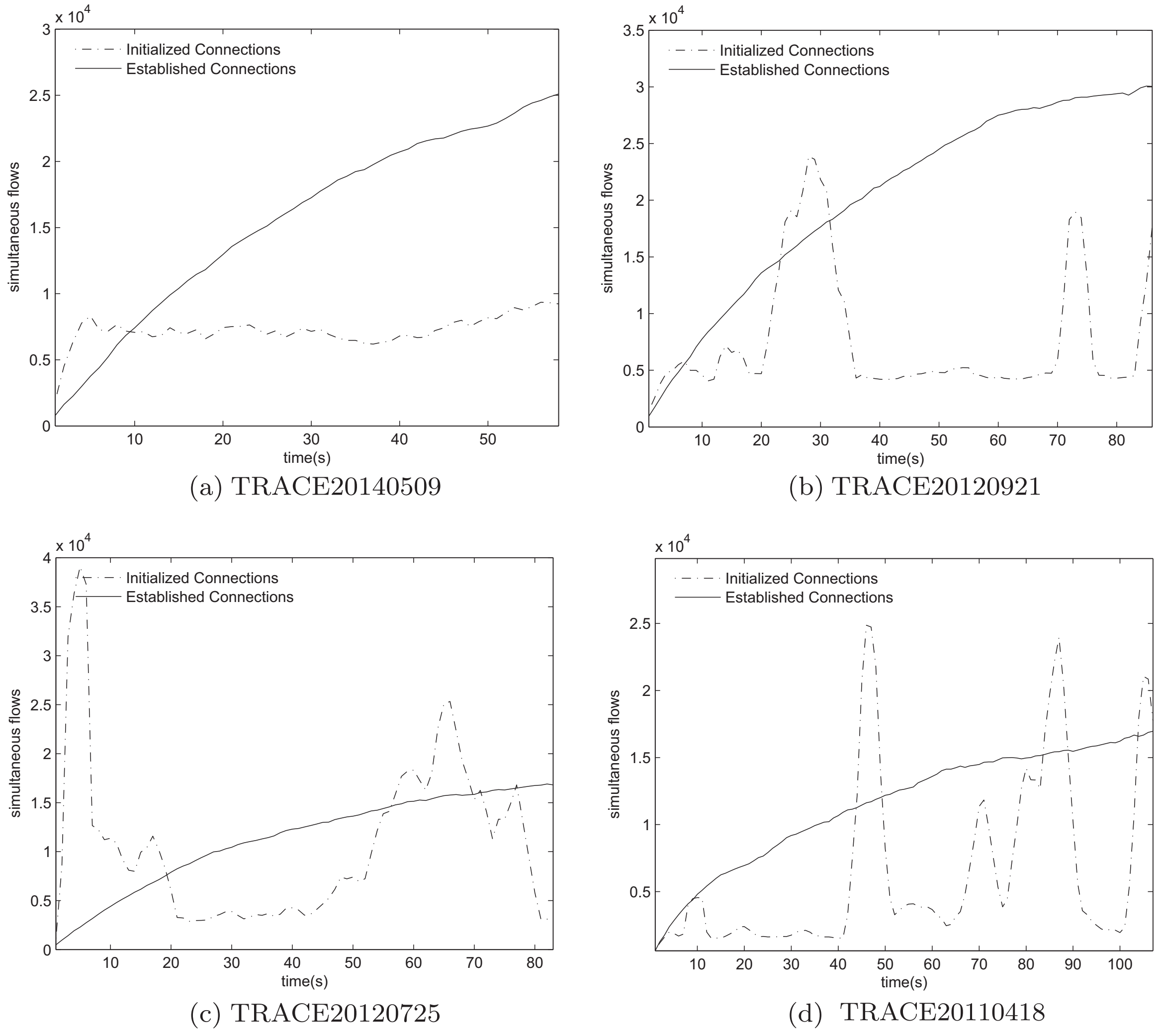


Fig. 4. The number of simultaneous flows in the traffic traces.

$$ASL_{DTP-SHT} = \frac{\upsilon}{\upsilon+1}(\upsilon\omega+1)\alpha + \frac{1}{\upsilon+1}\left(1+\frac{(1+\upsilon\omega)\alpha}{2}\right) = \frac{(\upsilon+0.5)(\upsilon\omega+1)\alpha+1}{\upsilon+1}. \tag{8}$$

Subsequently, we can deduce the packet distribution performance speedup of our proposed scheme DTP-CBF compared to the conventional one DTP-SHT in (9) with (7) and (8).

$$Speedup = \frac{ASL_{DTP-SHT}}{ASL_{DTP-CBF}} = \frac{(\upsilon+0.5)(\upsilon\omega+1)\alpha+1}{(p\upsilon^2\omega+0.5)\alpha+\upsilon+1}. \tag{9}$$

As in (9), the false positive error rate p is supposed as no more than 0.1 by setting the number of hash functions as k=6 and the ratio between the size of the counter array to the number of initial connections as $m/n = 8$. The load factor α is expected to take approximately 1, if the hash table length of the TCP connection table is configured closely to the number of TCP connections under normal conditions. The average number of packets within a connection ω is measured to fall around 16 for most of the time. Then, the speedup in (9) can be simplified in (10).

$$Speedup = \frac{16\upsilon^2+9\upsilon+1.5}{1.6\upsilon^2+\upsilon+1.5}. \tag{10}$$

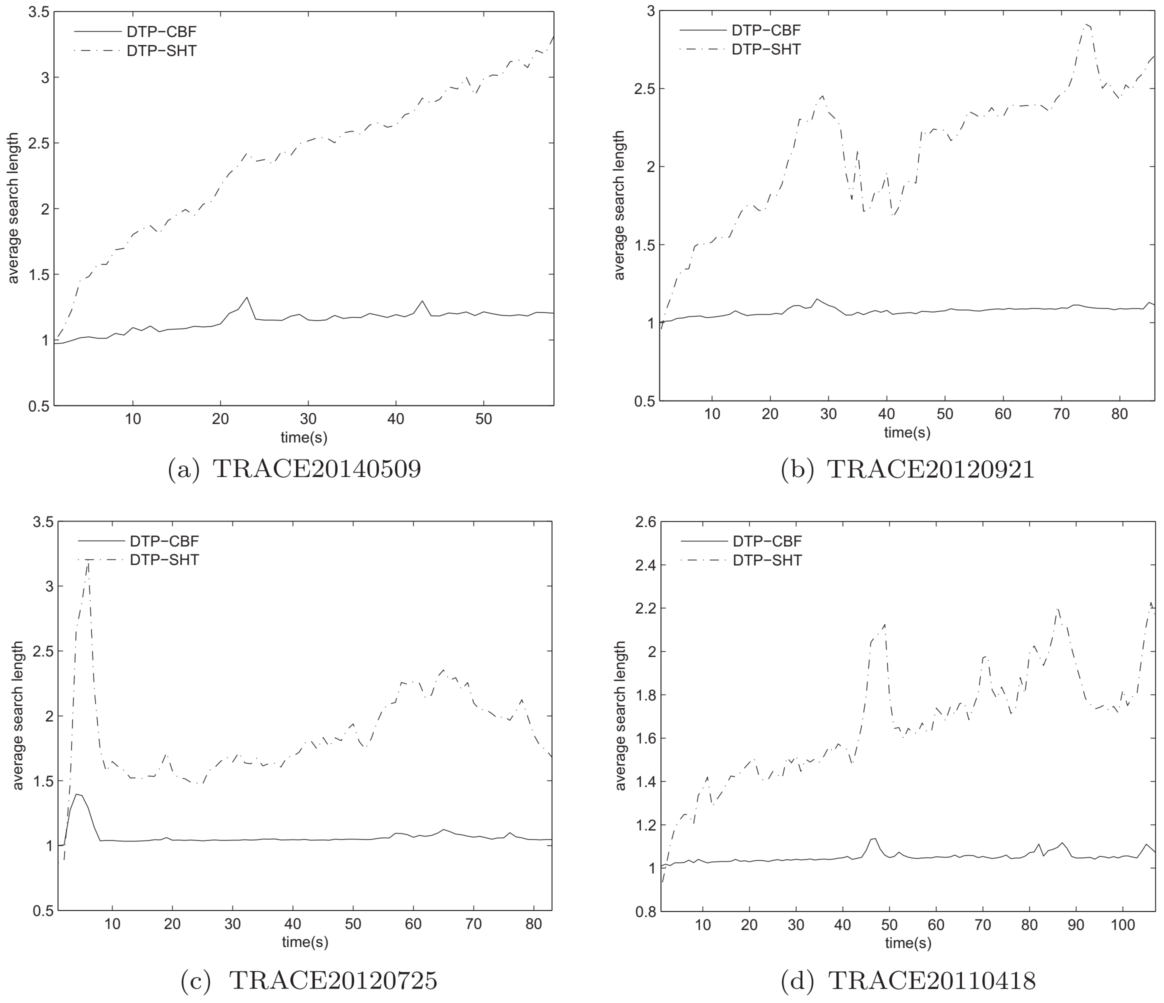
As seen from (10), the speedup chiefly depends on the ratio between the number of malicious packets and that of normal ones υ which reflects the intensity of malicious attacks. For example, the speedup will be respectively 4.17, 6.46 and 8.43, if υ=1/2, 1 and 2. In conclusion, the average search length of our proposed scheme will be much shorter than that of the conventional ones.

5. Experiments

This section evaluates the packet distribution performance of our proposed traffic partitioning scheme DTP-CBF especially under malicious attacks with real network traffic traces.

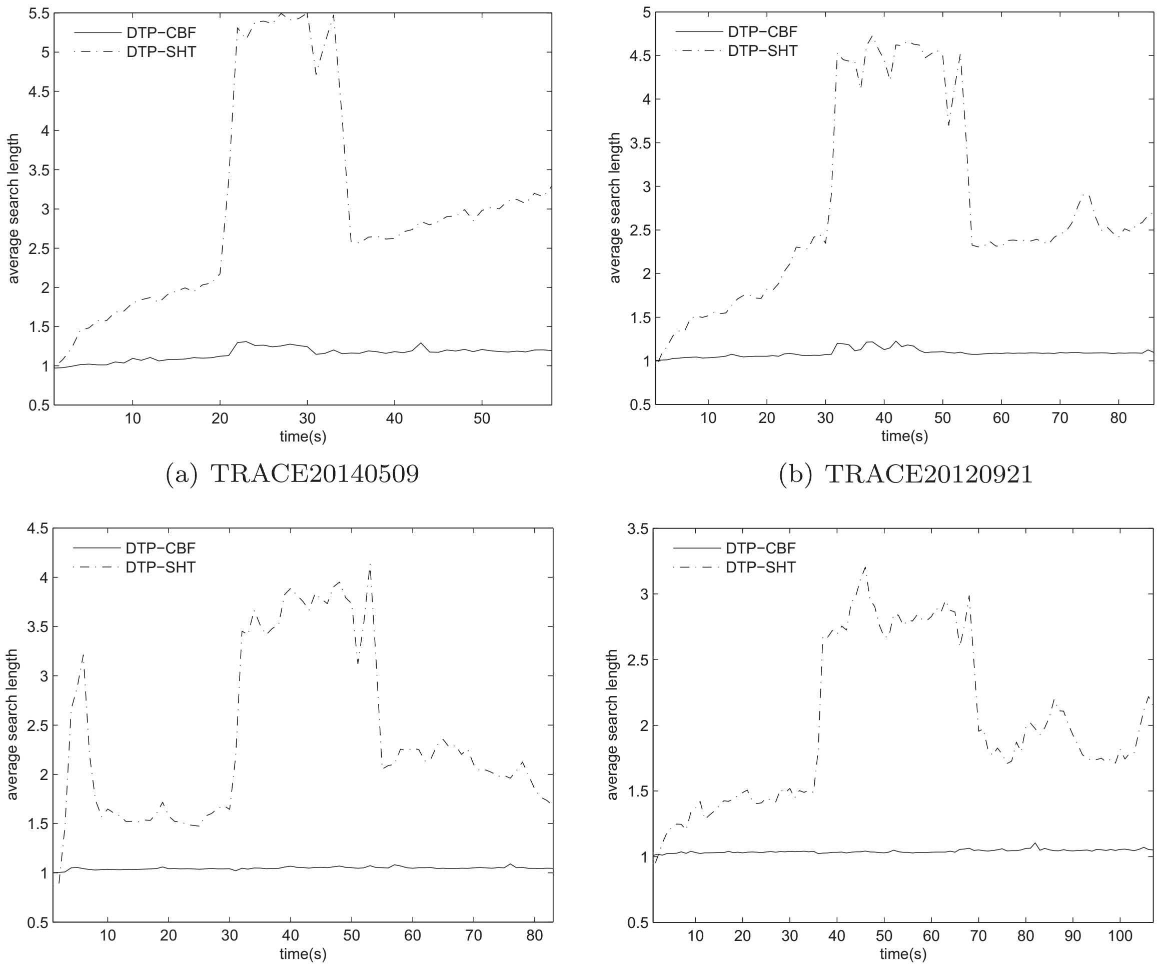
5.1. Traffic trace properties

For convenient multiple evaluation and comparison, we operate network traffic partitioning offline on traffic traces previously captured from backbone 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, our experiments use 4 traffic traces (Network traffic traces, 2015) collected from a 10 Gps main channel in the CERNET (Jiangsu), each of which



(a) TRACE20140509 (b) TRACE20120921
(c) TRACE20120725 (d) TRACE20110418

Fig. 5. The average search length of both schemes for the selected traffic traces.



(a) TRACE20140509 (b) TRACE20120921

(c) TRACE20120725 (d) TRACE20110418

Fig. 6. The ASLs with the packet hybrid ratio 1:2.

consists of 15,420,235 packets, and whose other properties are illustrated in Table 4.

Fig. 4 demonstrates the number of simultaneous connections in the traces, which has a significant impact on the packet distribution performance of dynamic traffic partitioning scheme. The number is counted by respectively setting 4 s and 60 s for the timeout interval of initial connections and established connections. As seen from Fig. 4, the number of established connections grows up linearly during the timeout interval and tends to stabilize after that, while the number of initial connections fluctuates in a wide range between 2K and 40K.

5.2. *Packet distribution performance*

With the above traffic traces, we contrast our proposed dynamic traffic partitioning scheme DTP-CBF with the conventional one in terms of packet distribution performance. The conventional scheme (Jiang et al., 2005, 2006; Li et al., 2002; Xiong et al., 2013) maintains simultaneous TCP connections in a single hash table, called DTP-SHT. As for TCP connection tables in both schemes, their hash table lengths are uniformly configured as 2^{14}. In addition, the counting bloom filter in our proposed scheme employs 6 hash functions, BOB, OAAT, TWMX, RS, Hsieh, and SBox (Henke et al., 2008; Molina et al., 2005).

5.2.1. *Experiment 1 (Normal Conditions)*

We first evaluate the packet distribution performance of both dynamic traffic partitioning schemes with the selected traffic traces under normal conditions. The counting bloom filter of the ICT table in our proposed scheme is configured in terms of $m = nk\log_2 e$, where m, n and k respectively represent the number of counters, initial connections and hash functions. As seen from Fig. 4, the number of initial connections n is approximately 8K on average. Then, the number of counters m is suitable to be set as 64K since the number of hash functions k equals to 6. With the above configurations, we run both dynamic traffic partitioning schemes on the TCP traffic in the selected traces, and get their average search length for each second in Fig. 5.

As seen from Fig. 5, our proposed DTP-CBF scheme outperforms the DTPSHT scheme in terms of average search length (ASL). In particular, our proposed scheme performs packet distribution with the ASLs about 1 almost at all times. In a striking contrast, the ASLs of the DTP-SHT scheme are much larger and always fluctuate with approximately 2 on average. This is chiefly attributed to the half-open connection separation mechanism by which most packets only need to look up one of the connection tables, i.e., the ICT table and the ECT table.

5.2.2. *Experiment 2 (Malicious Attacks)*

Then, we evaluate the packet distribution performance of both dynamic traffic partitioning schemes under malicious

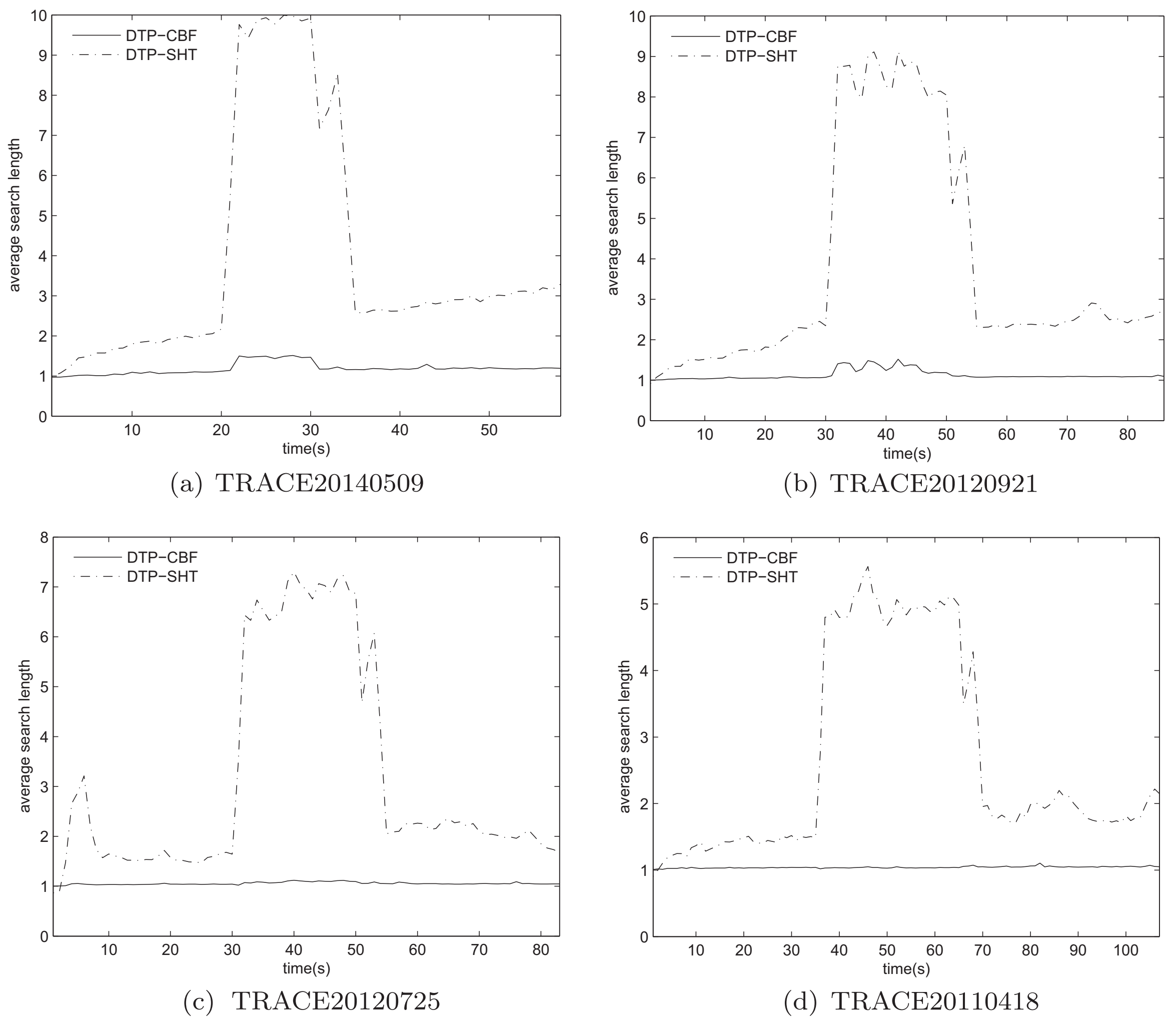


Fig. 7. The ASLs with the packet hybrid ratio 1:1.

attacks especially SYN flooding attacks. The attacks are simulated by inserting falsified SYN packets into normal TCP traffic in our selected traces. In particular, falsified packets are mixed with normal packets in different proportions to reflect diverse attack intensity.

As for the counting bloom filter, the number of its counters is configured in terms of $m = nklog_2e$, where the number of initial connections n, i.e. the size of the ICT table, is closely related to the attack intensity. The ICT table is dominated by unexpired false connections in the presence of malicious attacks, and its size 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 once SYN flooding attacks occur (Kim et al., 2005; Kang and Kim, 2003). The number of normal packets per second is around 128K in each traffic trace illustrated in Table 4. Besides, the number of hash functions k is fixed as 6. In summary, the number of counters m can be configured as 1M times as the packet hybrid ratio.

In our experiments, falsified SYN packets are inserted into TCP traffic in the 4 selected traces respectively during the 20th-30th second, 30th-50th second, 30th-50th second and 35th-65th second. Then we operate both dynamic traffic partitioning schemes on mixed traffic respectively with the packet hybrid ratio 1:2, 1:1 and 2:1, and calculate their average search length in Fig. 6, Fig. 7 and Fig. 8.

As seen from Fig. 6, Fig. 7 and Fig. 8, our proposed scheme DTP-CBF performs packet distribution with much shorter average search length than the conventional one DTP-SHT. In particular, the DTP-CBF scheme always takes steady short search lengths no matter how fierce the attacks are. In contrast, the DTP-SHT scheme degrades sharply in the presence of SYN flooding attacks. In conclusion, our proposed scheme provides much better robustness against SYN flooding attacks than the conventional one. This is attributed to the fact that all falsified SYN packets no longer need to search any flow tables as a result of the application of the counting bloom filter.

6. Conclusion and future work

Dynamic traffic partitioning schemes have been widely applied in high-speed network packet processing, which achieve excellent load balance effect at the price of heavy packet distribution overheads. This paper proposes a robust dynamic traffic partitioning scheme against malicious attacks, which builds the half-open connection separation mechanism to isolate false TCP connections in the ICT table, and applies counting bloom filters to the ICT table to defend against SYN flooding attacks. The experimental evaluations indicate that our proposed scheme performs much better than the conventional ones in terms of packet distribution performance. In particular, our proposed scheme performs packet

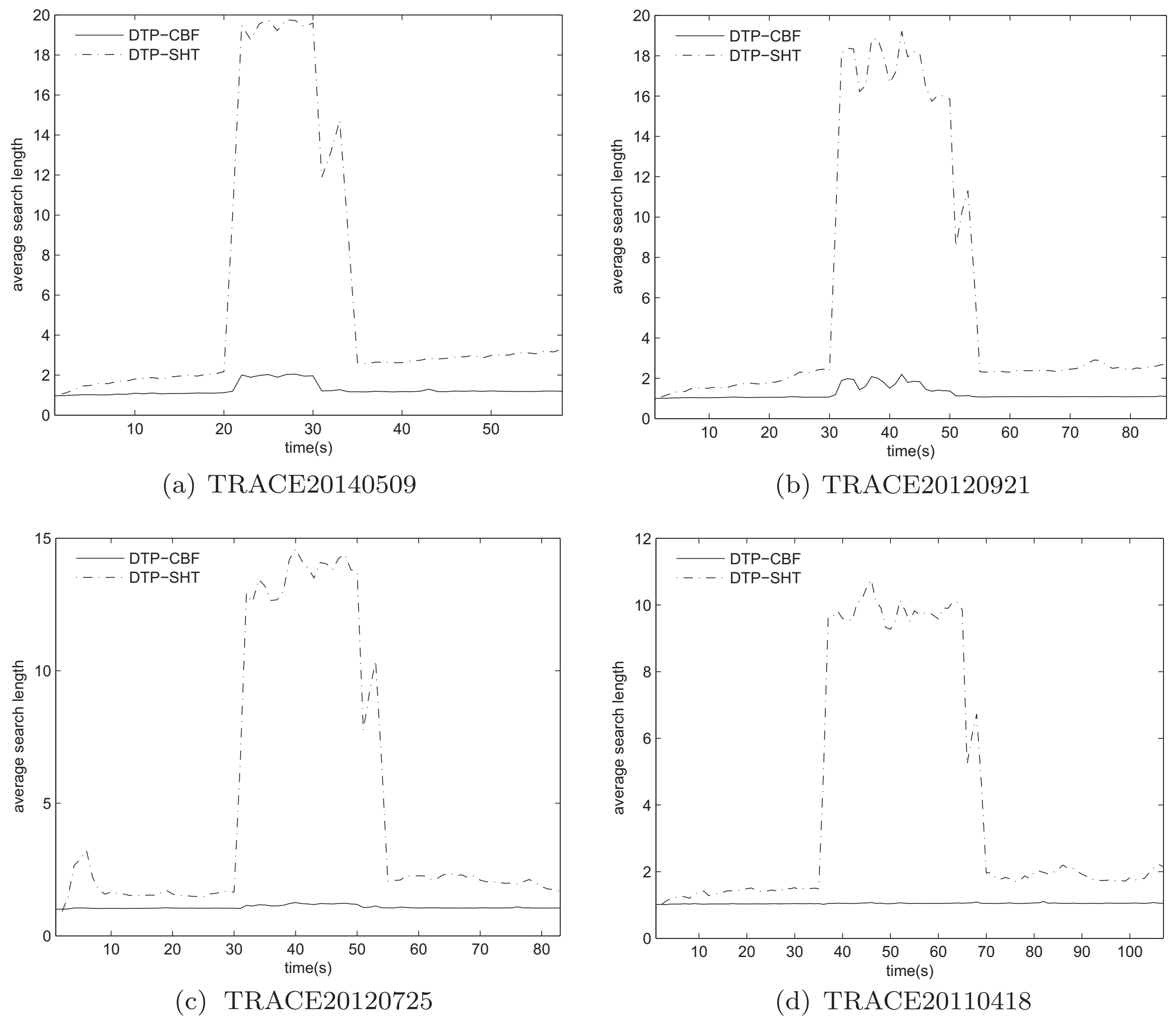


Fig. 8. The ASLs with the packet hybrid ratio 2:1.

distribution with the average search length steadily at about 1 even in the presence of SYN flooding attacks.

In our future work, more traffic traces from different high-speed network lines will be utilized to evaluate and validate our dynamic 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, are 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

Benetazzo, L., Narduzzi, C., Pegoraro, P. A. 2007. Internet Traffic Measurement: A Critical Study of Wavelet Analysis. IEEE Transactions on Instrumentation and Measurement, 56(3): 800-806.
Bloom, B., 1970. Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13 (7), 422–426.
Bredel, M., Bozakovy, Z., Barczyk, A., Newman, H., 2014. 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), pp. 213–214.
Cai, Y., Wu, B. Zhang, X., et al. 2014. Flow identification and characteristics mining from internet traffic with hadoop. In: Proceedings of International Conference on Computer, Information and Telecommunication Systems (CITS), Jeju, 1-5.
Cao, Z., Wang, Z., Zegura, E., 2000. Performance of hashing-based schemes for internet load balancing. In: Proceedings of IEEE International Conference on Computer Communications, Tel-Aviv, Israel, pp. 332–341.
Chen, Y., Lu, X., Shi, X., et al., 2008. A session-oriented adaptive load balancing algorithm. J. Softw. 19 (7), 1828–1836.
Cheng, G., Gong, J., Ding, W., 2003. Distributed sampling measurement model in a high speed network based on statistical analysis. Chin. J. Comput. 26 (10), 1266–1273.
Cohen, S., Matias, Y., 2003. Spectral bloom filters. In: Proceedings of 2003 ACM SIGMOD International Conference on Management of Data. ACM Press, New York, pp. 241–252.
Fall, K.R., Stevens, R.W., 2011. TCP/IP illustrated. In: The Protocols, 2nd edition, vol. 1. Addison-Wesley, Boston, Massachusetts.
Fan, L., Cao, P., Almeida, J., et al., 2000. Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Trans. Netw. 8 (3), 281–293.
Fulp, E.W., Farley, R.J., 2006. A function-parallel architecture for high-speed firewalls. In: Proceedings of IEEE International Conference on Communications, Istanbul, pp. 2213–2218.
Hakiri, A., Gokhale, A., Berthou, P., Schmidt, D.C., Gayraud, T., 2014. Software-defined networking: challenges and research opportunities for future internet. Comput. Netw. 75 (24), 453–471.
Henke, C., Schmoll, C., Zseby, T., 2008. Empirical evaluation of hash functions for multipoint measurements. ACM SIGCOMM Comput. Commun. Rev. 38 (3), 41–50.
Jiang, W., Song, H., Dai, Y., 2005. Real-time intrusion detection for high-speed networks. Comput. Secur. 24 (4), 287–294.
Jiang, W., Hao, S., Dai, Y., et al., 2006. Load balancing algorithm for high-speed network intrusion detection systems. J. Tsinghua Univ. (Sci. Technol.) 46 (1),

106–110.

Jo, J. Y., Kim, Y., 2004. Hash-based Internet traffic load balancing. In: Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI), Las Vegas, pp. 204–209.

Kang, I., Kim, H., 2003. Determining embryonic connection timeout in stateful inspection. In: Proceedings of IEEE International Conference on Communications, Anchorage, Alaska, pp. 458–462.

Kencl, L., Boudec, J.L., 2008. Adaptive load sharing for network processors. IEEE/ACM Trans. Netw. 16 (2), 293–306.

Kim, H., Kim, J., Kang, I., et al., 2005. Preventing session table explosion in packet inspection computers. IEEE Trans. Comput. 54 (2), 238–240.

Kirsch, A., Mitzenmacher, M., Varghese, G., 2010. Hash-based techniques for high-speed packet processing. Algorithms Next Gener. Netw. (Comput. Commun. Netw.) 2, 181–218.

Koerner, M., Kao, O., 2012. Multiple service load-balancing with OpenFlow. In: Proceedings of the IEEE 13th International Conference on High Performance Switching and Routing (HPSR), pp. 210–214.

Kreutz, D., Ramos, F.M.V., Verissimo, P., Rothenberg, E.C., Azodolmolky, S., Uhlig, S., 2015. Software-defined networking: a comprehensive survey. Proc. IEEE 103 (1), 14–76.

Lai, H., Huang, H., Xie, J., 2007. PABCS: a traffic partition algorithm for parallel intrusion detection. Chin. J. Comput. 30 (4), 555–562.

Li, X., Zhao, D., Zhao, H., et al., 2002. Research on application-based network intrusion detection system for high-speed network. J. China Inst. Commun. 23 (9), 1–7.

Manfredi, S., Oliviero, F., Romano, S.P., 2012. A distributed control law for load balancing in content delivery networks. IEEE/ACM Trans. Netw. 21 (1), 55–68.

Mohamed, N., Al-Jaroodi, J., Eid, A., 2013. A dual-direction technique for fast file downloads with dynamic load balancing in the cloud. J. Netw. Comput. Appl. 36 (4), 1116–1130.

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

Network traffic traces, 2015, ⟨http://iptas.edu.cn/src/system.php⟩.

Patel, A., Taghavi, M., Bakhtiyari, K., Junior, J.C., 2013. An intrusion detection and prevention system in cloud computing: a systematic review. J. Netw. Comput. Appl. 36 (1), 25–41.

Park, J. S., Lee, J. Y., Lee, S. B., 2000. Internet traffic measurement and analysis in a high speed network environment: Workload and flow characteristics. Journal of Communications and Networks, 2(3): 287-296.

Qi, Y., Xu, B., He, F., Yang, B., Yu, J.M., Li, J., 2007. 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, pp. 17–26.

Shi, W., MacGregor, M.H., Gburzynski, P., 2005. Load balancing for parallel forwarding. IEEE/ACM Trans. Netw. 13 (4), 790–801.

Sun, Q., Zhang, D., Gao, P., et al., 2004. Study of parallel ids load balancing algorithm. Mini-micro Syst. 25 (12), 2215–2217.

Thompson, K., Miller, G.J., Wilder, R., 1997. Wide-area internet traffic patterns and characteristics. IEEE Netw. 11, 10–23.

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

Williamson, C., 2001. Internet traffic measurement. IEEE Internet Comput. 5 (6), 70–74.

Wolf, T., Cai, Y., Kelly, P. Gong, W., 2007. Stochastic Sampling for Internet Traffic Measurement. In: Proceedings of IEEE Global Internet Symposium, Anchorage, 31-36.

Xiong, B., Xiao, H., Long, M., et al., 2013. Dynamic partitioning of high-speed network traffic with flow level. Mini-micro Syst. 34 (5), 945–950.

Zhang, Z.H., 2011. Distribute and match - the DM switch for high speed packet switching networks. In: Proceedings of IEEE Global Telecommunications Conference, pp. 1–6.

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