Computer Communications 32 (2009) 1893–1901

Contents lists available at ScienceDirect

Computer Communications

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

A TCAM-based solution for integrated traffic anomaly detection and policy filtering

Zhijun Wang [a,*], Hao Che [b], Jiannong Cao [a], Jingshan Wang [a]

[a] Department of Computing, The Hong Kong Polytechnic University, Hung Hum, Hong Kong
[b] Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX 76019, USA

ARTICLE INFO

Article history:
Received 12 November 2007
Received in revised form 20 June 2009
Accepted 17 July 2009
Available online 28 July 2009

Keywords:
Flow anomaly detection
Two-dimensional matching
TCAM coprocessor

ABSTRACT

The survivability of the future Internet is largely dependent on whether it will be able to successfully address both security and performance issues facing the Internet. On one hand, the Internet becomes more and more vulnerable due to fast spreading malicious attacks. On the other hand, it is under great stress to meet ever growing/changing application demands while having to sustain multi-gigabit forwarding performance. In this paper, we propose a Ternary Content Addressable Memory (TCAM) coprocessor based solution for high speed, integrated TCP flow anomaly detection and policy filtering. The attacking packets with spoofed source IP addresses are detected through two-dimensional (2D) matching. The key features of the solution are: (1) setting flag bits in TCAM action code to support various packet treatments; (2) managing TCP flow state in pair to do 2D matching. We evaluate the solution's ability to detect TCP-based flooding attacks based on real-world-trace simulations. The results show that the proposed solution can match up OC-192 line rate. The possible modifications of the solution for the detection of low rate TCP-targeted attacks are also discussed.

1. Introduction

The fast spreading malicious attacks make the Internet more and more vulnerable, while the ever growing application demands require more and more types of Internet services. The future Internet has to address both security and performance issues to survive.

Distributed Denial of Service (DDoS) attacks [11] are the major threats to the Internet. In DDoS attacks, attackers send a large amount of attacking packets using spoofed source IP addresses to a victim server which eventually runs out of its resources and degrades the performance of legitimate packets. One effective way to defend against DDoS attacks is IP traceback [3,6,9,16,17,20,21,23,24]. Using IP traceback, the attackers can be identified and punished through tracing their physical locations. However, effectively identifying attackers is difficult due to the stateless property of Internet routers/switchs. Hence it is important and challenging to do IP traceback.

The fast growing application demands need the network to provide various types of services. To support differential services, different packets may need to be treated differently based on, e.g., quality-of-service requirements or other policies. To this end, packet classification [2,15,18,25] based on a set of policy filtering rules must be performed in a router interface to identify the needed treatment of individual packets. Traditional policy filters [2,15,18,25] treat each packet individually, and does not attempt to associate the packet with other packets belonging to the same flow. Flow classification is a stateful packet classification, generally known as packet classification, which tracks the flow state by identifying every packet in every flow. Packet/flow classification has long been identified as the most critical data path function, creating potential bottlenecks for high speed packet forwarding.

To remove these potential bottlenecks, various algorithmic and hardware approaches have been developed, attempting to meet the targeted performance for efficiently PF. However, implementing different solutions to execute multiple tasks is very costly and sometimes even infeasible due to various resource constraints. Hence, it is both technologically and economically important to develop integrated solutions for PF and Content Filtering (CF), matching multi-gigabit line rate or even higher.

The traditional approach to enable security functions is generally separated from the approach that implements typical packet forwarding functions. For example, hash-based IP traceback [23] is generally implemented using dedicated chips for computing hash functions. Packet classification is typically performed as part of the packet forwarding functions in a router interface card, e.g., using a network processor and its associated Ternary Content Addressable Memory (TCAM) coprocessor [5,27]. A TCAM coprocessor contains self-addressable rules which map to different memory addresses in an associated memory (normally an SRAM) containing the corresponding actions. A rule matching in a TCAM is performed for all the rules in parallel. Each parallel matching is done at a time. Due to its high speed performance, the TCAM coprocessor is widely used as packet classifier in industry. How-

* Corresponding author. Tel.: +852 2766 7277.
E-mail address: cszjwang@comp.polyu.edu.hk (Z. Wang).

0140-3664/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2009.07.016

ever, the separated solutions add the complexity and integration costs to the next-generation Internet design. Hence, it is of both technological and economical importance to develop integrated solutions to enable security functions and high speed forwarding, matching multiple gigabit line rate.

In this paper, we propose a TCAM coprocessor based solution for high speed, integrated TCP traffic anomaly detection and policy filtering. The TCP-based DDoS attacks using spoofed source IP addresses are detected in the edge router through two-dimensional (2D) matching [8]. 2D matching means a normal TCP flow generated from one end host to another should have a corresponding flow from the other direction. The key features of the solution are: (1) setting flag bits in TCAM action code to support various packet treatments in the network processor and the local CPU; (2) managing TCP flow state in pair to do 2D matching. In the solution, when a TCP flow has not been matched after a period of time T_{alm}, the flow is considered to have high probability to be an attacking flow. Hence an alarm message composed of the flow identities is sent to the destination server, which in turn can use the information to do IP traceback. Based on the real Internet traffic analysis, the proposed solution requires about 5 Mbits TCAM memory to support OC-192 line rate for the integrated tasks. Such TCAM is available in today's market. We also discuss how to handle TCAM table overflow and analyze the solution's performance in case of table overflow. The simulations based on the real world traffic traces are conducted to evaluate the performance on the detection of TCP-based flooding attacks. The results show that the proposed solution can handle OC-192 line rate. The modifications on the proposed solution for the detection of low rate TCP-targeted attacks are also discussed.

The proposed solution focuses on the detection of TCP-based attacks.[1] The non-TCP-based attacks can be defended by other existing solutions such as Probabilistic Packet Marking [9,16,21,24] and Internet Control Protocol Message (ICMP)-based [3] IP traceback solutions. The real traffic measurement shows that the TCP traffic constitutes 80% of the total traffic (see Section 4), and the TCP-based attacks account for more than 80% of over all attacks [11]. Hence monitoring non-TCP packets for defense against DDoS attacks is much less challenging than monitoring TCP packets.

The rest of the paper is organized as follows: Section 2 describes the TCP traffic anomaly detection through 2D matching. The details of the integrated solution is presented in Section 3. The performance of the proposed solution is evaluated by simulations in Section 4. Section 5 discusses how to modify the solution to detect low rate TCP attacks. Section 6 briefly describes the related work. Finally, Section 7 concludes the paper and discusses some future work.

2. TCP traffic anomaly detection through 2D matching

In this section, we first give the needed definitions and then discuss how to detect anomalous TCP flows through two-dimensional (2D) matching.

A *flow* is a set of packets which have the same identity. The identity is extracted from the packet header. In this paper, the following five tuples: source IP address (SIP), destination IP address (DIP), source port number (SPN), destination port number (DPN), and protocol (PRO) are used as the *flow identity*. In other words, a flow is uniquely determined by the five tuples <SIP, DIP, SPN, DPN, PRO>.

TCP is a two-way communication protocol. A normal TCP flow generated from one end host (e.g., *A*) to another (e.g., *B*) should have a corresponding flow from the other direction (i.e., from *B* to *A*). Fig. 1 shows a general Internet architecture. Assume host *A*

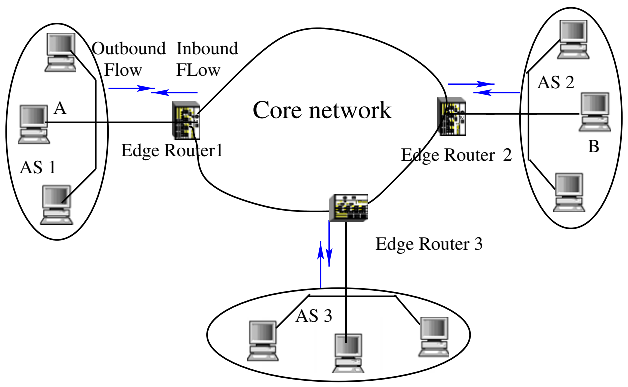

Fig. 1. Internet architecture.

in Autonomous System 1 (AS1) sends a SYN packet to host *B* located in AS2 to initiate a TCP session. After receiving the SYN packet, host *B* sends a SYN + ACK packet back to host *A* to establish the session. In this case, the edge router 1 can detect both flows coming from AS1 (called *outbound flow*) and into AS1 (called *inbound flow*). For an outbound flow with flow identity <SIP, DIP, SPN, DPN, PRO>, the corresponding inbound flow identity is <DIP, SIP, DPN, SPN, PRO>. The feature of an outbound flow having a corresponding inbound flow is called two-dimensional (2D) matching [8]. An outbound (inbound) flow is called an *unmatched flow* if no corresponding inbound (outbound) flow arrives within a period of time T_{alm}. An inbound (outbound) flow is called the *matching flow* of its outbound (inbound) flow.

2D matching can be effectively applied to detect TCP-based attacks using spoofed IP addresses. For attacking packets using spoofed source IP addresses, the responding packets are routed to the spoofed IP addresses which may be different from the original AS. Thus the edge router at the attackers' AS may only detect the outbound flow, and hence an unmatched flow is detected. Based on these observations, one can do 2D matching at the edge routers for TCP traffic anomaly detection. When an unmatched flow is detected, the router sends an alarm message (e.g., ICMP message) including the flow identity to the destination for possible IP traceback.

The most popular TCP-based DDoS attacks are TCP SYN and RESET flooding attacks. In these attacks, the SYN or RESET flag bit is set. To detect these attacks, we only need to maintain all the flows start with SYN and RESET packets. However, there are other types of TCP-based attacks [11] which have ACK bit set or no flag bit set. Hence, any TCP packet can be an attack packet. In our solution, if a packet does not belong to any existing flow, the packet is considered to be a new flow and will be monitored in the flow table to allow 2D matching.

Except the attacking packets, unmatched flows may be caused by: (1) the destination server is down; (2) the destination server has changed its IP address, but a cache entry of the old server IP address is still in the domain name server (DNS). In these cases, the destination is unreachable and the flows sent to the destination server can be viewed abnormally.

In the following sections, we will present the details on how to integrate 2D matching and policy filtering using TCAM coprocessors.

3. Integrated TCP traffic anomaly detection and packet classification

This section first gives a brief review of policy filtering using a network processor and its TCAM coprocessor, and then presents the details of the proposed solution.

[1] In fact, any attack with two-dimensional matching feature can be measured through the proposed solution. For example, ICMP echo reply attacks. For simplicity, we focus on the TCP-based attacks here.

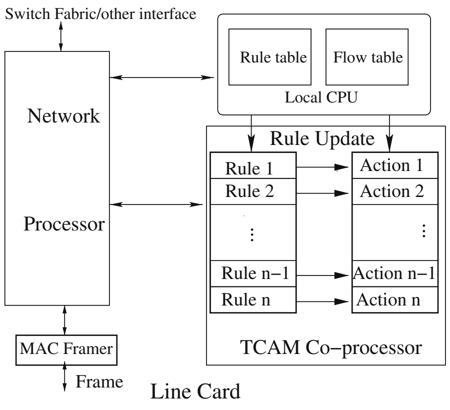

Fig. 2. TCAM coprocessor architecture.

flag	forward	action	flow index	free
1–bit	1–bit	8–bit	22–bit	2–bit

(a) Format of action code

flag	FIN	ACK	Tlocation	Timer
2–bit	1–bit	1–bit	22–bit	32–bit

(b) Format of flow table in local CPU

Fig. 3. Data structures.

3.1. Architecture of network processor using TCAM coprocessor

Packet classification (e.g., policy filtering, IP forwarding table lookup) is one of the most critical data path functions in high speed packet forwarding. TCAM coprocessors are widely used as packet classifiers in today's industry. Fig. 2 shows a system architecture of a network processor using a TCAM coprocessor [27] for packet classification. A TCAM coprocessor stores self-addressable rules which map to different memory addresses in an associated memory (normally an SRAM) containing the corresponding actions.

In particular, a typical rule for policy filtering is composed of 104-bit five tuples: <SIP, DIP, SPN, DPN, PRO>, same as the flow identity. The rules are usually arranged in an ordered list, with lower memory locations having higher matching priorities. When a packet arrives at the network processor, a search key composed of the same set of five tuples, extracted from the packet header is passed to the TCAM coprocessor for lookup. The action code in the associated memory corresponding to the matched rule with the highest match priority is returned to the network processor. There is an identical copy of the rule table in the local CPU in charge of rule management. The rule update in TCAM is done through the interface between the TCAM coprocessor and the local CPU. In the proposed solution, when a new flow is detected, its identity serves as a new rule to be added to the rule table, meaning that the rule table for flow classification is combined with the policy filtering table for packet classification. In addition, a flow table in the local CPU is introduced to store active TCP flow states for 2D matching.

A general rule usually has some wildcarded bits in some tuples, whereas the identity of a specific flow has no wildcarded bits in any tuple. For example, a policy filtering rule may be: <x.x.x.x, x.x.x.x, 128–256, 80, 6>. Here 'x' represents a wildcarded byte. A flow identity may be <1.2.3.4, 5.6.7.8, 1028, 80, 6>. A flow identity has higher matching priority than a policy filtering rule. So a flow identify must be located at a lower TCAM memory address than a policy filtering rule.

3.2. Data structure for integrated solution

The policy filtering rules and the flow identities share the same format. Hence, the integrated approach does not require any modification to the rule table format in TCAM. To support IP traceback, however, the action codes need to be extended to allow flow detection. Before giving the details of the solution, we first present the data structures of the action code and flow table entry.

Fig. 3(a) shows the format of the action code. The action code is set to 32 bits in length such that the code can be returned in one clock cycle through a 32-bit interface bus. The action code includes one flag bit which indicates if a rule is a policy filtering rule (value 0) or a flow identity (value 1). One forward bit is used to indicate if the action code needs to be passed (value 1) to the local CPU or not (value 0). The following 8 bits indicate the forwarding action associated with the policy filtering, such as the best effort forwarding, dropping the packet and so on. Eight bits can express 256 different actions which are enough to include all possible forwarding actions in today's Internet. The last 22 bits are the flow index which specifies the location of a flow state in the flow table located in the local CPU. Twenty-two bits can represents 4 million different entries. For a policy filtering rule, all index bits are set to 0. The last two bits are free bits, and always set to zero. In the following, we use *a*-*b*-*x*-*y*-00 to represent an action code. Here *a* and *b* represent the binary values of the first two bits, respectively. *x* the decimal value of the action, and *z* the decimal value of the flow index. For example, (0-0-5-0-00) represents the action code of a policy filtering rule with action 5; and (1-1-6-234-00) is the action code of a flow identity with action 6, it locates at entry 234 in the flow table, and the forward bit set means the action code needs to be passed to the local CPU.

Fig. 3(b) gives the data structure of an entry in the flow table. The first 2 bits are flags. The bits 00 indicate an empty entry; 01 an unmatched existing flow; 10 an expected flow; and 11 a matched existing flow. FIN and ACK bits are used to terminate a pair of completed flows. Tlocation is the flow location in the TCAM rule table. Timer is used to trigger an event. There are three timers: T_{alm}, T_{idl} and T_{rmv}. T_{alm} detects an unmatched flow, an alarm message is triggered if a flow has not been matched after T_{alm} time. T_{idl} is used to check if a matched flow is still active. If a pair of flows are not terminated after T_{idl} time, the forward bits corresponding to them are set to check if they are still active. T_{rmv} is used to remove incompletely terminated flows. The values of the three thresholds are related to the response time of TCP open, close and reset. They should be longer than the response time of most TCP open, close and reset. The longer threshold values result in fewer false alarms but cost more resources to maintain the unmatched, idle and completed flows. Similarly, *ab*-*c*-*d*-*x*-*y* is used to express a flow entry. *a*, *b*, *c* and *d* are the binary values of the first four bits, *x* and *y* are decimal values of Tlocation and timer, respectively.

To do 2D matching, the flow entries in the flow table are managed in pair. When a new flow arrives, the index is always set to an even number, and the expected matching flow has the index equal to the even number plus one. An index list is used to manage the available indices. The number of entries in the flow table is set to be the number of total TCAM entries allocated for flow identities. Initially, all even indices are on the list. When a new flow is detected, an index on the list is removed and assigned to the new flow. When a pair of flows are finished and removed from the flow table, the corresponding even index is returned to the list.

The rule table in the local CPU is in charge of the TCAM rule table management. It stores the same rules (including both policy filtering rules and flow identities) as those in the TCAM rule table. The management of the two rule tables are the same, hence in the following of the paper, the rule table refers to the rule tables in both local CPU and TCAM.

3.3. Description of the integrated solution

In the proposed solution, the local CPU processes packet flows at per flow level while the TCAM coprocessor processes packet flows at per packet level. When a packet arrives at the network processor, the search key composed of five tuples extracted from the packet header is passed to the TCAM coprocessor. The action code (a-b-x-y-00) corresponding to the matched rule with highest matching priority is returned back to the network processor. The network processor forwards the packet based on the action code value. For a non-TCP packet, no extra processing is introduced. For a TCP packet, the action code *a-b-x-y-fk* is passed to the local CPU for the following three cases: (1) the packet belongs to a new flow (i.e., $a = 0$); (2) the packet belongs to an existing flow but the forward bit is set (i.e., $a = 1$ and $b = 1$); (3) the packet with FIN bit set (i.e., $a = 1, \ b = 0$ and FIN bit = 1). The two free bits *f* and *k* are set to be the bit values of FIN and ACK bits in the packet, respectively. In the case of the arrival packet belonging to a new flow, the flow identity is also passed to the local CPU for process.

The local CPU is in charge of adding new flows, testing flow activity, removing completed and inactive flows, and triggering unmatched alarms. Now we describe how the local CPU handles different packets and timer timeouts.

3.3.1. Packet in new flow

When a packet belonging to a new flow arrives, if no free entry is available in the rule table, the action code and the flow identity are simply dropped. Otherwise, the new flow (F) and its expected matching flow (E) are added to both the flow and rule tables. In the flow table, suppose the index of F is IN, then the index of E is $IN + 1$. The first two flag bits are 01 at entry IN implying that F is an existing unmatched flow and 10 at entry $IN + 1$ indicating that E is an expected matching flow. The flag bits in the action codes of both flows are set to 1, and the forward bit corresponding to E is set to 1. Hence the action code of the upcoming packet in E will be passed to the local CPU to do 2D matching. The forward bit of F is 0, implying that the upcoming packets in F do not need to be processed in the local CPU. The timer T_{alm} is set for F in the flow table.

3.3.2. Packet in expected flow

If a packet belonging to an expected flow E arrives, the two flag bits in the flow entries for both E and its matching flow F are set to 11 indicating that the two flows are matched. The forward bit in the action code corresponding to E is reset to 0. Then the upcoming packets in the pair of flows will not be processed in the local CPU. Hence a timer T_{idl} is set for both flows E and F. If no FIN bit is detected within T_{idl} time, the forward bits for both flows will be set to test if they are still active.

3.3.3. Packet in matched flow

The forward bits for a pair of matched flows are set after T_{idl} expires. This means that the action codes of the upcoming packets in the pair of flows will be passed to the local CPU. If such a packet arrives, it implies that the pair of flows are still active. Hence the timer T_{idl} is set again for the next check. Then the forward bits in the action codes for the pair of flows are reset to 0.

3.3.4. Packet with FIN and/or ACK bit set

If a packet of F with FIN bit set arrives, the FIN bit for its entry in the flow table is set. The forward bits in the action code for the pair of flows are also set, meaning that the action codes of the upcoming packets in the pair of flows will be processed in the local CPU. If a FIN + ACK packet of F comes, the FIN bit is set for entry F. If its matching flow E has FIN bit set, the packet is an acknowledgement packet of the FIN packet in E, and hence the ACK bit in the entry of E is set. If an ACK packet of E comes after both FIN bits set, this acknowledges FIN packet in F, hence the ACK bit in the flow entry of F is set. If the FIN and ACK bits are set for both flows, this implies that the two flows have been completed and hence to be removed from the flow and rule tables. In order to remove incompletely terminated (without FIN and/or ACK packets) flows, when the FIN bit is set for an entry, timer T_{rmv} is also set for the pair of flows. If no more packet arrives within T_{rmv} time, the flow pair is forced to be removed.

3.3.5. Timer timeout

If the timer T_{alm} times out, it implies that an unmatched flow is detected, an alarm message including the flow identity is generated and sent to the destination server for possible IP traceback. If the timer T_{idl} is triggered, it indicates that no FIN packet in the pair of flows arrives in the past T_{idl} time. However, that does not ensure that the flow pairs are still active, because some flows may be terminated incompletely. Hence it needs to check if they are still active, so the forward bits in their action codes are set, and the timer T_{rmv} is also set. If no packet for the pair of flows arrives within T_{rmv} time, the flows are considered to be inactive and removed from the tables. In this case, if some packets in the removed flows come later, they will be treated as new flows.

3.4. Illustration of packet processing

Now we use one example to illustrate the TCP packet processing. Assume a SYN packet (pk_1) of flow f_1 which has identity <1.2.3.4, 5.6.7.8, 80, 1028, 6> arrives. The search key composed of the flow identity is matched against all the rules in the TCAM rule table. Assume that the search key matches the rule x.x.x.x-5.x.x.x-80-1028-6 with the highest matching priority as shown in Fig. 4(a). The action code 0-0-3-0-00 is returned. The flag 0 indicates the packet belonging to a new flow and hence the network processor passes the action code and flow identity to the local CPU. Two indices are assigned for f_1 and its expected matching flow e_1 which has identity <5.6.7.8, 1.2.3.4, 1028, 80, 6>. Two flows are added to both

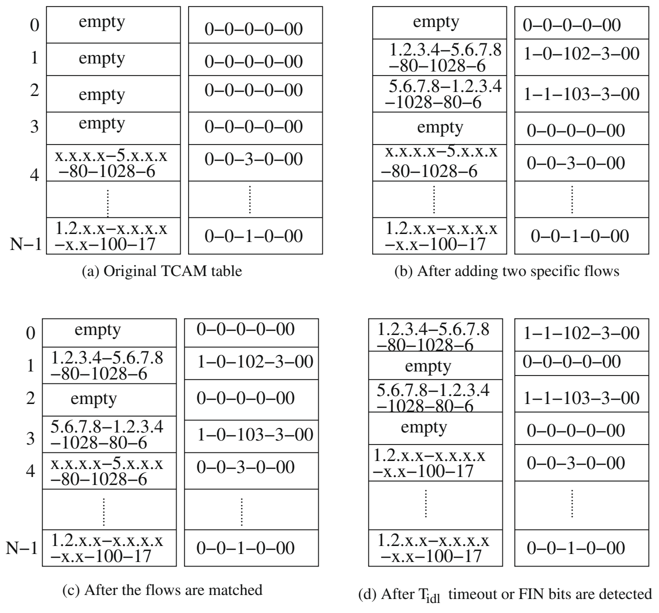
0
1
2
3
4
N−1
empty
x.x.x.x−5.x.x.x −80−1028−6
1.2.x.x−x.x.x.x −x.x−100−17
0−0−0−0−00
0−0−3−0−00
0−0−1−0−00
(a) Original TCAM table
1.2.3.4−5.6.7.8 −80−1028−6
5.6.7.8−1.2.3.4 −1028−80−6
1−0−102−3−00
1−1−103−3−00
(b) After adding two specific flows
1−0−103−3−00
(c) After the flows are matched
1−1−102−3−00
(d) After T_idl timeout or FIN bits are detected

Fig. 4. TCAM rule table update.

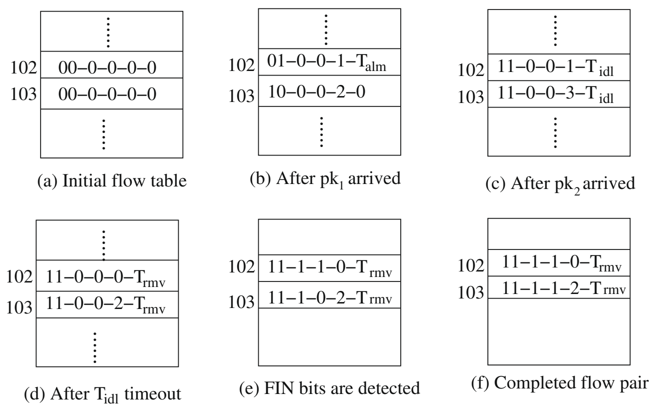

Fig. 5. Flow table update.

the flow (at 102 and 103) and rule (at 1 and 2) tables. The timer T_{alm} is set for f_1 for triggering alarm. The entries in the flow table at 102 and 103 are 01-0-0-1-*Talm* and 10-0-0-2-0, respectively, as shown in Fig. 5(b). In the rule table, entry 1 has rule f_1 with action code 1-0-102-3-00 and entry 2 has flow e_1 with 1-1-103-3-00 shown in Fig. 4(b).

Now assume the SYN + ACK packet (pk_2) of e_1 arrives. The search key matches the rule 5.6.7.8-1.2.3.4-1028-80-6, the action code 1-1-103-3-00 is returned. The forward bit 1 indicates the action needs to be passed to the local CPU. The local CPU checks the action code, and updates the flag bits at both entries 102 and 103 to 11 which indicates that flows f_1 and e_1 are matched flows. The timer T_{idl} is set for both flows. The forward bit for e_1 is reset to 0. The TCAM entry of e_1 is updated (see [27] for consistent rule table update) to a new location as shown in Fig. 4(c), and the flow table is update as shown in Fig. 5(c).

Now, let us assume no packet with FIN bit set in f_1 and e_1 arrives before T_{idl} expires. Then when T_{idl} expires, the forward bits of f_1 and e_1 are set to test if the flows are still active. Now the updated rule table is shown in Fig. 4(d), and the updated flow table is presented in Fig. 5(d) with entries 11-0-0-0-T_{rmv} and 11-0-0-2-T_{rmv} at indices 102 and 103.

Now assume a packet (pk_3) in flow f_1 with ACK bit set arrives, the action code 1-1-102-3-00 is returned and the action code 1-1-102-3-01 is passed to the local CPU. This packet indicates that the pair of flows are still active, and they will be tested again after T_{idl} time. Hence timer T_{idl} is set again, and the forward bits are reset to 0 for both flows. Now the flow table is given in Fig. 5(c), and the rule table is shown in Fig. 4(b).

Suppose a packet (pk_4) with FIN bit set in f_1 arrives, the returned action code is 1-0-102-3-00. Due to the FIN bit set, the action code 1-0-102-3-10 is passed to the local CPU. The local CPU sets the FIN bit in the flow table and the forward bit in the action code for flow f_1. Now a packet (pk_5) with FIN + ACK bits set in e_1 comes, the action code 1-0-103-3-11 is passed to the local CPU, the FIN bit of e_1 is set. While the ACK bit in f_1 is set due to its FIN bit set. The flow table is updated as shown in Fig. 5(e), and the rule table is updated as shown in Fig. 4(d).

Finally, assume an ACK packet in f_1 comes, the action code 1-1-102-3-01 is passed to the local CPU. The ACK bit of e_1 is set (as in Fig. 5(f)) and hence both flows are removed from both flow and rule tables.

3.5. Computation load on local CPU

The local CPU processes packet at per flow level. For each flow, the first, FIN and final ACK packets are processed. In addition, 1 packet needs to be processed every T_{idl} time. The real traffic measurement at OC-192 (see Section 4) shows that the average new TCP flows are about 5 K per second, and the concurrent active flows are usually less than 50 K. That means a flow lasting less than 10 s on average. If T_{idl} is set to 5 s, then each flow has about 5 packets or 0.2 packets per second to be processed. The local CPU needs to process about 20 K packets per second which is not difficult to be handled by a 100 MHz CPU.

3.6. TCAM rule update

The TCAM rule table update is through the interface between the local CPU and the TCAM coprocessor. A consistent rule update algorithm [27] which can update the rule table without interruption of TCAM lookup process is used. A flow identity has higher matching priority than that of policy filtering rules and hence it is added to a TCAM memory location lower than the policy rules. All the flow identities can be stored independently, because a packet cannot match more than one flow identity simultaneously. We keep all the empty entries above the general rules so that for each flow identity addition or deletion, no movement is needed for other rules. Adding one rule takes 5 (a rule plus the action code have 104 + 32 bits, (104 + 32)/32 < 5) clock cycles by assuming a 32-bit interface bus. Deleting a rule only takes 1 clock cycle by reset the valid bit of the entry [27]. To update a flow identity, a flow identity is first written and validated in a new location and then the flow identity in the old location is deleted. It takes one rule write and one rule deletion. To process 20 K packets, maximum 40 K TCAM writes/updates (assume the action code changes before and after the packet being processed) are needed. 40 K TCAM writes and deletion only take about 240 K clock cycles, this is a very small portion of the processing time for a 100 MHz CPU.

3.7. TCAM table overflow

There are two critical issues using TCAM for packet/flow classification. One is the TCAM lookup speed, and the other is the memory space. The proposed solution does not introduce any extra TCAM lookups. Hence we only focus on the discussion of TCAM memory space. TCAM coprocessors have limited memory storage. The maximum TCAM memory in today's market is 18 Mbits [4]. A TCAM is usually shared by multiple tables such as longest prefix matching and policy filtering tables. Hence the TCAM memory storage capacity is a critical issue for the proposed solution. A critical concern is that if the TCAM table cannot store all the concurrent active flows, does the solution still work well? In the following, we discuss the performance impacts in case of table overflow.

When the local CPU detects a new flow, it checks if there are a pair of free entries in the TCAM rule table. The flow state is monitored only if a pair of free entries are available. That means an attacking packet may not be monitored immediately in case of table overflow, and hence it may not detect attacks with single attacking packet. But it can still detect the attacks with a large amount attacking packets. In case of table overflow, each coming packet in an unmonitored flow has some probability to be monitored, hence it takes some time to monitor a packet from an attack. The following theorem gives the average time an attack to be monitored.

Theorem 1. *Assume a TCAM rule table has N entries. There are $n_c > N$ number of concurrent active normal TCP flows, each flow has n_f packets per second, and the TCAM accepts n_s number of new flows per second. Then an attacker sending n_a attacking packets per second can be monitored in $\frac{(n_c-N)n_f-n_s}{n_a n_s}$ time on average.*

Proof. The total number of arriving packets per second is $n_c n_f$, among these packets, Nn_f packets belong to the flows monitored in the TCAM rule table. The number of packets belonging to the flows which are not monitored is $(n_c - N)n_f$, so each packet has

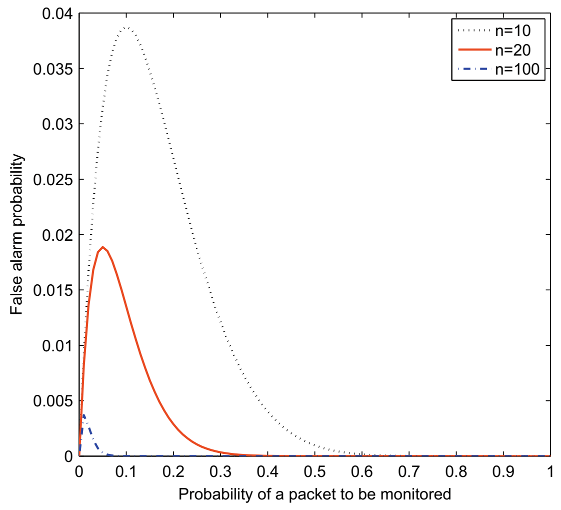

Fig. 6. False alarm probability.

probability $p = \frac{n_s}{(n_c - N)n_f}$ to be monitored. For the attack, each attacking packet has p probability to be monitored, the packet inter arrival time is $1/n_a$. So the average time an attack to be monitored is $t_p = \sum_{k=1}^{\infty}\left[(1-p)^{k-1}p \times \frac{k-1}{n_a}\right] = \frac{1-p}{pn_a} = \frac{(n_c-N)n_f - n_s}{n_a n_s}$. □

Theorem 1 shows that the detection time of an attack is inversely proportional to the attacking rate (i.e., the number of attacking packets per second). The time to be monitored of an attack with 1 packet per second is 10 times of an attack with 10 packets per second on average. This is verified in the simulations (see next section).

In case of table overflow, every packet in a flow can be the first packet to be monitored. For a pair of TCP flows, if the last packet of these two flows is monitored, the flow will never be matched because both flows are completed. In this case, a normal TCP flow is considered to be an attacking packet, and a false alarm will be generated. The following theorem gives the false alarm probability of a pair of flows with total n number of packets.

Theorem 2. *Assume there are a total of n packets in a pair of flows, each packet has probability p to be monitored, then the false alarm probability is* $p_{false} = (1-p)^{n-1}p$.

Proof. The false alarm happens if the first $n-1$ packets are not monitored and the last one is monitored. So the probability of a false alarm is $p_{false} = (1-p)^{n-1}p$. □

The maximum false alarm probability occurs at $p = 1/n$ for a pair of flows with a total of n packets. Fig. 6 shows the falsely alarmed probability for the pair of flows with 10, 20, and 100 packets. The false alarm probability for the pair of flows with 100 packets is almost zero in all the probability range. For the pair of flows with 10 packets, the maximum false alarm probability is about 0.04 at $p = 1/10$, the maximum probability is dropped to 0.02 when the number of packets in the pair of flows reaches 20. The probability of a packet in a new flow to be monitored depends on the number of rule entries in the TCAM. If the TCAM can monitor all the concurrent active flows, $p = 1$, and hence no false alarm occurs.

4. Performance evaluation

This section evaluates the performance of the proposed solution in the case of table overflow. If the TCAM rule table can monitor all the concurrent active TCP flows, the system can detect attacks even with single attacking packet, while the normal TCP flows are not falsely alarmed. In the case of TCAM rule table overflow, it takes some time to monitor an attack; a normal flow may be falsely alarmed; and some packets belonging to un-monitored flows are dropped by the local CPU. These are quantitatively studied by simulations. In the simulations, four performance metrics: the average time (t_m^A) an attack to be monitored, the number (n_{false}) of false alarm flows per second, the number of TCAM writes per second, and the number of un-monitored packets per second are measured. t_m^A is the average time difference between the arrival time of the first attacking packet in an attack and the arrival time of the attacking packet in that attack being monitored. The time an attack to be monitored is zero if the first attacking packet in an attack is monitored.

Table 1
Statistics of real traffic trace.

	Trace 1	Trace 2
Total number of packet	12,746,894	14,494,880
Number of TCP packets	9,217,812	11,138,762
Average number of new TCP flows per second	4381	5181
Average number of concurrent active flows	19,423	24,476
Maximum number of concurrent active flows	21,030	26,718

The simulations are conducted based on the real router traces downloaded from Abilene-IV Trace Data [10]. These traces are captured from OC-192 backbone Abilene router to and from Kansas city. Each trace includes 90-s traffic on both inbound and outbound links. We have tested more than 10 traces, and selected one trace with average statistics (Trace 1) and one with maximum number of packets (Trace 2). Table 1 shows the basic statistics of the two traces. In both traces, the TCP packets account for more than 80% of the total packets. We also note that the number of new TCP flows per second is about 5 K, and the number of maximum concurrent active flows is less than 30 K which map to less than 4 Mbits TCAM memory (assume each rule takes 128 bits in the TCAM rule table). We suggest to use 4 Mbits TCAM rule table to store flow identities to avoid table overflow. Usually there are thousands of policy filtering rules which takes less than 1 Mbits TCAM memory. Hence a TCAM rule table with 5 Mbits memory is enough to do the integrated tasks in OC-192 line rate in today's Internet.

We evaluate the performance of the proposed solution in case of TCAM table overflow by set a small TCAM rule table. In the simulations, the number of entries in the TCAM rule table varies from 5 K (640 Kbits) to 25 K (3.2 Mbits). The attacking packets are generated starting at the 15th second. Three attacking rates (R): 20, 100 and 500 packets per second are simulated. From our trace data, more than 99% TCP SYN and FIN packets can be acknowledged within 1 s. Hence the three threshold can be set to longer than 1 s. In the simulations, a pair of matched flows are tested every 5 s ($T_{idl} = 5$) to check if they are still active. A pair of flows are removed from all the three tables if they are completely finished or inactive for a time period longer than 5 s since it was tested, i.e., $T_{rmv} = 5$. An unmatched flow is considered to be an attacking flow if it exits for a time period longer than 10 s, i.e., $T_{alm} = 10$.

Figs. 7 and 8 show the average time an attack to be monitored (t_m^A) varying with the TCAM rule table size. t_m^A in Trace 1 (2) is within 11 (20) seconds in the entire range of rule table sizes. For rule table with 5 K entries, t_m^A in Trace 1 (2) is about 11 (19.5), 2.2 (4), and 0.4 (0.8) seconds for $R = 20, 100, 500$, respectively. t_m^A in Trace 2 is greater than that in Trace 1, because Trace 2 has more new flows per second and more concurrent active flows. From the results, we note that t_m^A is inversely proportional to the attacking rate, verifying Theorem 1. So for $R = 1$, t_m^A is about 20 times of that for $R = 20$, i.e., about 220 (400) seconds. t_m^A reduces fast as the

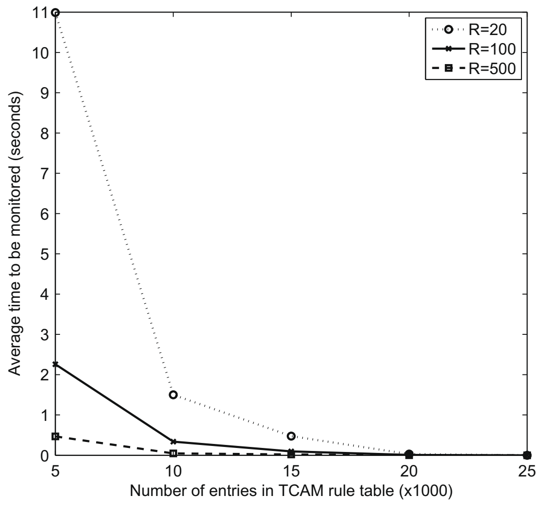

Fig. 7. Average time an attack to be monitored in Trace 1.

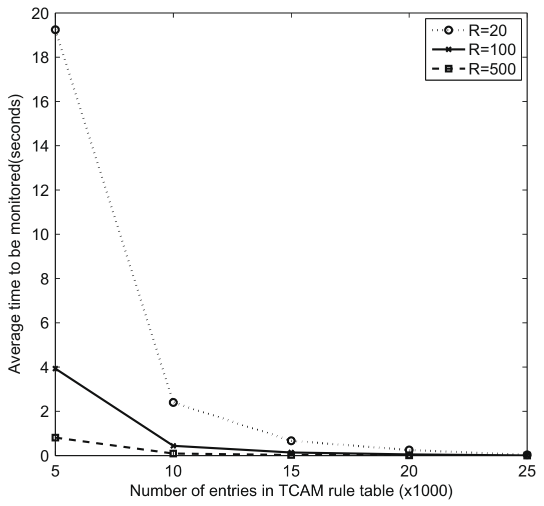

Fig. 8. Average time an attack to be monitored in Trace 2.

number of rule entries increases. This is because the TCAM rule table can accept more rules and hence each packet has higher probability to be monitored. These results show that the proposed solution can quickly detect the attacks even if the attacking rate is low and the TCAM rule table is small.

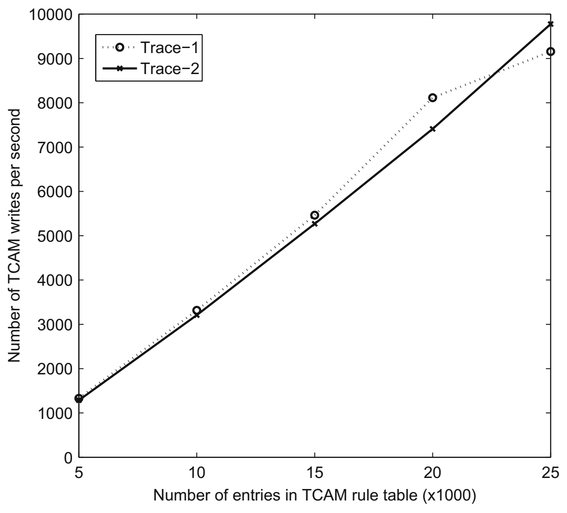

Fig. 9. Number of TCAM writes per second.

Table 2
Number of TCP packets belonging to un-monitored flows per second.

TCAM size	5 K	10 K	15 K	20 K	25 K
Trace 1	19,294	13,655	9155	1565	0
Trace 2	23,160	15,613	11,279	7189	1726

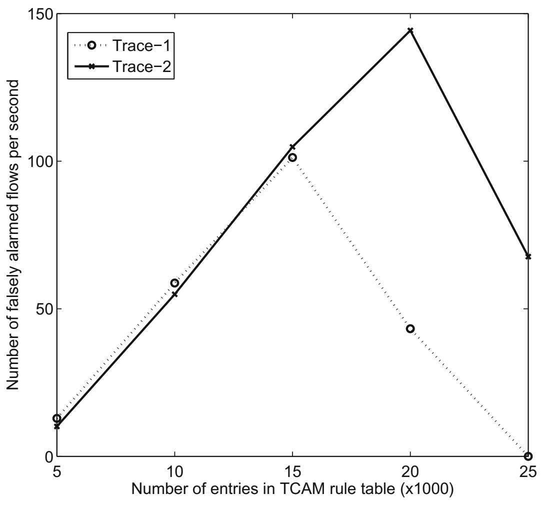

Fig. 10. Number of falsely alarmed flows per second.

Fig. 9 shows the number of TCAM writes including adding new flows and update flow entries per second in both traces. The number of writes per second increases from about 1.5 K to about 10 K as the number of TCAM rule entries increases from 5 K to 25 K. This is due to the fact that more flows are added to the rule table when the TCAM table size increases. When the number of TCAM entries increases from 20 K to 25 K, the number of writes per second in Trace 1 increases slowly. This is because almost all TCP flows are monitored in the rule table at 20 K, and hence only a few more TCP flows can be added by further increasing the TCAM table size.

The network processor passes the action code of all packets belonging to un-monitored flows to the local CPU. Some of these packets are simply dropped due to lack of free rule entries. Table 2 shows the number of the dropped packets per second at various rule table sizes. The average number of TCP packets per second in Trace 1 (2) is 102,420 (123,764), the number of packets per second in un-monitored flows are only about 19 K (23 K) even using a very small TCAM rule table. It reduces quickly as the table size increases.

The number of false alarm flows per second (n_{false}) is given in Fig. 10. n_{false} increases as the rule table size increases when the table size is small. It then decreases as the table size increases when the table size is over a certain value. The maximum n_{false} in Trace 1 (2) is about 100 (150) at the rule table size 15 K (20 K). When the number of rule entries is 5 K, n_{false} is about 10. As shown in Theorem 2, n_{false} depends on the probability (p) of a packet to be monitored. p increases as the rule table size increases, and results in the first increasing and then decreasing behavior.

From these results, we suggest to use either a large (4 Mbits or above) or small (640 Kbits) TCAM rule table to accommodate flow identities. For a large one, no table overflow exists, and hence no falsely alarmed flows. For a small one, the number of falsely alarmed flows is small and also the number of TCAM writes is small, but the attack detection time is long. For a middle sized TCAM, when the number of alarms is high, a small monitoring probability can be set for each new flow even if there are some free entries. In other words, a new flow may not be monitored even if the free entries are available in the rule table. This can reduce the number of falsely alarmed flows.

Through these results, we conclude that the proposed integrated solution can effectively perform the two tasks even using a small TCAM. The proposed solution can be implemented in router/switch which use TCAM coprocessors in its router interface cards.

5. Detection of low rate TCP-targeted DoS attacks

In this section, we discuss how to modify the proposed solution to detect low rate TCP-targeted DoS attacks. Low rate TCP-targeted DoS attack [12] exploits the fixed minimum TCP retransmission Time Out (RTO) and periodically sends a short burst of data to a targeted router/server to force all the affected TCP flows to enter the retransmission timeout state. Due to the low traffic rate, this type of attacks is more difficult to be detected than the flooding based DoS attacks. Randomizing TCP RTO [28] is an effective way to defend such type of attacks. But it may reduce the TCP performance when no attack exists.

Some detection methods through monitoring the traffic flow patterns have been proposed [19,22,26] to against low rate TCP-targeted attacks. The key issue in these methods is how to effectively monitor the per flow state to determine if a flow shows the periodic burst pattern. Our proposed solution can be easily modified to detect such type of attacks. To detect low rate TCP attacks, when a new flow arrives, all the upcoming packets are monitored (with forward bit set), and the packet inter arrival time is recorded. The traffic patterns can be identified through analyzing the packet inter arrival time. Whenever a flow is identified as a regular flow, the flow is monitored at the flow level, i.e., the flow is periodically tested. In the case of no table overflow, all TCP flows are monitored in the flow table, and hence the packet arrival patterns can be easily captured for low rate TCP-targeted attack detection. Moreover, due to the ability of measuring 2D flows, more sophistical detection schemes [19] can be implemented such as monitoring the traffic volumes in both directions. In the case of table overflow, our proposed solution is still valid but the attacking flow may be detected with some delay. As discussed in the previous section, if the attack has tens of attacking packets per second, the detection time is only tens of seconds even if a TCAM rule table is small.

6. Related work

The DDoS attacks are the major threat to today's Internet. One effective way to defend against such attacks is to identify and punish the attackers through tracing their physical locations. IP traceback schemes have been extensively studied in the past decade. These schemes includes statistical filtering, hop-by-hop tracing, ICMP messaging based, hash-based and probabilistic packet marking. Statistical filtering [6,17] drops most likely attacking packets based on the statistics of packet header information such as IP address, port number, protocol type, etc. Hop-by-hop tracing scheme [13] uses a pattern-based scheme to track in progress attacks. ICMP messaging based scheme [3] sends additional ICMP packet to the destination for path reconstruction. Hash-based solution [23] computes and stores a Bloom filter digest of every packet for IP traceback. Probabilistic packet marking solution [9,16,21,24] marks each packet with partial path information probabilistically. The attacking path is reconstructed using the marking information extracted from a large number of packets.

There are other schemes for DDoS attacks and/or traffic anomaly detection. Lakhina et al. [14] proposed a general method to diagnose traffic anomaly by measuring traffic volume. Fan et al. [8] designed a bloom filter array for traffic anomaly detection through 2D matching. In [1], a DDoS defense system based on the packet score has been developed. The bad packets with score less than the threshold are discarded.

TCAM has been widely used for longest prefix matching and policy filtering table matching [15,25] in industry. Except the longest prefix matching and policy filtering, another promising application of TCAM coprocessors is high-speed signature matching for intrusion detection [7]. A distributed TCAM coprocessor architecture [29] for longest prefix matching that matches OC-768 line rate was proposed. However, there is no application of TCAM coprocessor to enable integrated security and packet classification tasks.

7. Conclusions

In this paper, we propose an integrated solution for TCP-based traffic anomaly detection and policy filtering based on TCAM coprocessors. The DDoS attacks using spoofed source IP address are detected through two-dimensional (2D) matching. The key features of the proposed solution are: (1) setting flag bits in TCAM action code to support various packet treatments; (2) managing TCP flow state in pair to do 2D matching. The performance of the proposed solution has been analyzed and tested by simulation based on the real world traffic traces. The results show that the proposed solution can handle OC-192 line rate. The possible modifications of the proposed solution to defend low rate TCP-targeted attacks are also discussed. Using the inbound and outbound IP prefix matching will be studied for UDP-based traffic anomaly detection.

Acknowledgement

This work is supported by the UGC of Hong Kong under the CERG Grant PolyU 5293/06E.

References

[1] P.E. Ayres, H. Sun, H.J. Chao, A high-speed PacketScore DDoS defense system, IEEE Journal on Selected Areas in Communications 24 (10) (2006) 1864–1876.
[2] F. Baboesu, G. Varghese, Scalable packet classification, in: Proceedings of ACM SIGCOMM, 2001.
[3] S. Bellovin, ICMP traceback messages, Work in Progress, Internet Draft draft-bellovin-itrace-00.txt, 2000.
[4] Cypress Ayama 10K/20K NSE Series TCAM Products. Available from: <http://www.cypress.com>.
[5] H. Che, Z. Wang, K. Zheng, B. Liu, DRES: dynamic range encoding scheme for TCAM coprocessors, IEEE Transactions on Computers 57 (7) (2008) 902–915.
[6] Z. Duan, X. Yuan, J. Chandrashekar, Constructing inter-domain packet filters to control IP spoofing based on BGP updates, in: Proceedings of IEEE INFOCOM, 2006.
[7] Fang Yu, Randy H. Katz, T.V. Lakshman, Gigabit rate packet pattern matching with TCAM, in: IEEE ICNP, 2004.
[8] J. Fan, D. Wu, K. Lu, A. Nucci, Design of bloom filter array for network anomaly detection, in: Proceedings of IEEE GLOBECOM, 2006.
[9] M.T. Goodrich, Efficient packet marking for large-scale IP traceback, in: Proceedings of ACM Conference on Computer and Communications Security (CCS), 2002.
[10] http://www.pma.nlanr.net/Specical/ipsl4.html.
[11] A. Hussain, J. Heidemann, C. Papadopoulos, A framework for classifying denial of service attacks, in: ACM SIGCOMM, 2003.
[12] A. Kuzmanovic, E.W. Knightly, Low-rate TCP-targeted denial of service attacks, in: ACM SIGCOMM, 2003.
[13] J. Joannidis, S.M. Bellovin, Implementing pushback: router-ased defense against DDoD attacks, in: Network and Distributed System Security Symposium, 2002.
[14] A. Lakhina, M. Crovella, C. Diot, Diagnosing network-wide traffic anomalies, in: Proceedings of ACM SIGCOMM, 2004.
[15] K. Lakshminarayanan, A. Rangarajan, S. Venkatachary, Algorithms for advanced packet classification with ternary CAMs, in: Proceedings of ACM SIGCOM, 2005.

[16] J. Li, M. Sung, J. Xu, L. Li, Large-scale IP traceback in high-speed internet: practical techniques and theoretical foundation, in: Proceedings of IEEE Symposium on Security and Privacy, 2004.
[17] Q. Li, E. Chang, M. Chan, On the effectiveness of DDoS attacks on statistical filtering, in: IEEE INFOCOM, 2005.
[18] J. Lunteren, T. Engbersen, Fast and scalable packet classification, IEEE Journal on Selected Areas in Communications 21 (4) (2003) 560–571.
[19] X. Luo, E. Chan, R. Chang, Vanguard: a new detection scheme for a class of TCP-targeted denial-of-service attacks, in: IEEE/IFIP Network Operations & Management Symposium, 2006.
[20] T. Peng, C. Leckie, K. Ramamohanarao, Protection from distributed denial of service attacks using history-based IP filtering, in: Proceedings of IEEE International Conference on Communications, 2003.
[21] S. Savage, D. Wetherall, A. Karlin, T. Anderson, Network support for IP traceback, IEEE/ACM Transactions on Networking 9 (3) (2001) 226–237.
[22] A. Shevtekar, K. Anatharam, N. Ansari, Low rate TCP denial-of-service attack detection at edge routers, IEEE Communication Letters 9 (4) (2005) 363–365.
[23] A.C. Snoeren, C. Partridge, L.A. Sanchez, C.E. Jones, Hash-based IP traceback, in: Proceedings of ACM SIGCOMM, 2001.
[24] D.X. Song, A. Perrig, Advanced and authenticated marking schemes for IP traceback, in: Proceedings of IEEE INFOCOM, 2001.
[25] E. Spitznagel, D. Taylor, J. Turner, Packet classification using extended TCAMs, in: Proceedings of International Conference of Network Protocol (ICNP), September, 2003.
[26] H. Sun, J. Lui, D. Yau, Defending against low-rate TCP attacks: dynamic detection and protection, in: IEEE INCP, 2004.
[27] Z. Wang, H. Che, M. Kumar, S. Das, CoPTUA: consistent policy table update algorithm for TCAM without locking, IEEE Transactions on Computers 53 (12) (2004) 1602–1614.
[28] G. Yang, M. Gerla, M. Sanadidi, Defense against low-rate TCP-targeted denial-of-service attacks, in: IEEE Symposium on Computers and Communications (ISCC), 2004.
[29] K. Zheng, C. Hu, H. Lu, B. Liu, A TCAM-based distributed parallel IP lookup scheme and performance analysis, ACM/IEEE Transaction on Networking 14 (4) (2006) 863–875.