Telecommun Syst (2018) 68:67–78
https://doi.org/10.1007/s11235-017-0375-3

Reducing false rate packet recognition using Dual Counting Bloom Filter

Ivica Dodig[1] · **Vlado Sruk**[2] · **Davor Cafuta**[1]

Published online: 24 August 2017

Abstract Distributed Denial of Service (DDoS) attacks are a serious threat to Internet security. A lot of research effort focuses on having detection and prevention methods on the victim server side or source side. The Bloom filter is a space-efficient data structure used to support pattern matching problems. The filter is utilised in network applications for deep packet inspection of headers and contents and also looks for predefined strings to detect irregularities. In intrusion detection systems, the accuracy of pattern matching algorithms is crucial for dependable detection of matching pairs, and its complexity usually poses a critical performance bottleneck. In this paper, we will propose a novel Dual Counting Bloom Filter (DCBF) data structure to decrease false detection of matching packets applicable for the $SACK^2$ algorithm. A theoretical evaluation will determine the false rate probability of detection and requirements for increased memory. The proposed approach significantly reduces the false rate compared to previously published results. The results indicate that the increased complexity of the DCBF does not affect efficient implementation of hardware for embedded systems that are resource constrained. The experimental evaluation was performed using extensive simulations based on real Internet traces of a wide area network link, and it was subsequently proved that DCBF significantly reduces the false rate.

Keywords DDoS · SYN flooding · Hash-based detection · Bloom filter

✉ Ivica Dodig
ivica.dodig@tvz.hr

[1] Zagreb University of Applied Sciences, Vrbik 8, Zagreb, Croatia

[2] Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, Zagreb, Croatia

1 Introduction

The Internet has evolved and greatly improved communication and business. The increase in the number of users and business enterprises on the Internet also gives rise to increased illegitimate activity. These activities are known as security threats. The most common security threats are network attacks which are used to obstruct normal communication [1]. The consequences of such attacks are potentially devastating and may vary from an attack against a single user to complete network obstruction where the whole network can be made unusable for a longer period of time. This type of attack occurs across a wide range of protocols [2].

Network attacks focused on communication disruption are called Distributed Denial of Service (DDoS) attacks. DDoS attacks prevent users from communicating with the attacked computer network because the victim network or servers fail to provide normal services. DDoS attacks occur when the system receives too much traffic for the server to buffer, causing it to slow down and eventually halt operations. The effectiveness of the attack is proportional to the number of computers performing the attack. Different mechanisms have been developed for early detection and prevention of DDoS attacks at the router level within a network infrastructure [3–5]. They mostly relate to indications of increasing trends in such attacks [1,6]. Currently, considerable efforts are underway to develop better detection algorithms and to identify DDoS at the earliest stage of an attack.

Recent research has suggested creating new communication models. One such example is the information-centric model where routing is based on packet data as opposed to router tables on nodes. The underlying nodes and hosts become more-or-less impartial, possibly to the extent that there are no longer persistent addresses and names. This new proposal does resolve some issues (e.g. improved accuracy).

However, certain vulnerabilities enabling possible DDoS attacks still remain [7].

DDoS detection schemes can be classified into schemes based on the router data structure, statistical analysis of packet flows and use of artificial intelligence [8]. Router data structure schemes are mostly applicable to detection concepts for embedded systems. Minimal processing power and memory requirements are a key factor for successfully implementing this detection algorithm on an Intrusion Detection System (IDS) [9]. An IDS can be implemented as a stand alone embedded system or even as a part of the main network router [10]. The DDoS attack should be detected as close as possible to the source to reduce the overall network load. In the event of a DDoS attack, this is impossible because the sources are distributed over the Internet. Thus, detection has to take place at the destination network. There is a possibility of positioning one IDS node before the destination network. An example would be an ISP point of entry, where the ISP manages the local network. Figure 1 shows the position of an IDS system within a network.

In router data structure schemes, detection is based on identifying disruption in the ratio of control packets. A DDoS attack disrupts the normal ratio of control packet types as it floods the victim with unanswered requests [3]. An example of a DDoS attack is TCP SYN flooding and is most prevalent in successful attacks [11,12]. It involves sending SYN packets to a server that are never confirmed due to receiving the SYN/ACK from the attacking server. In such cases, the control ACK packet, establishment packets and four-way finish control packets are missing.

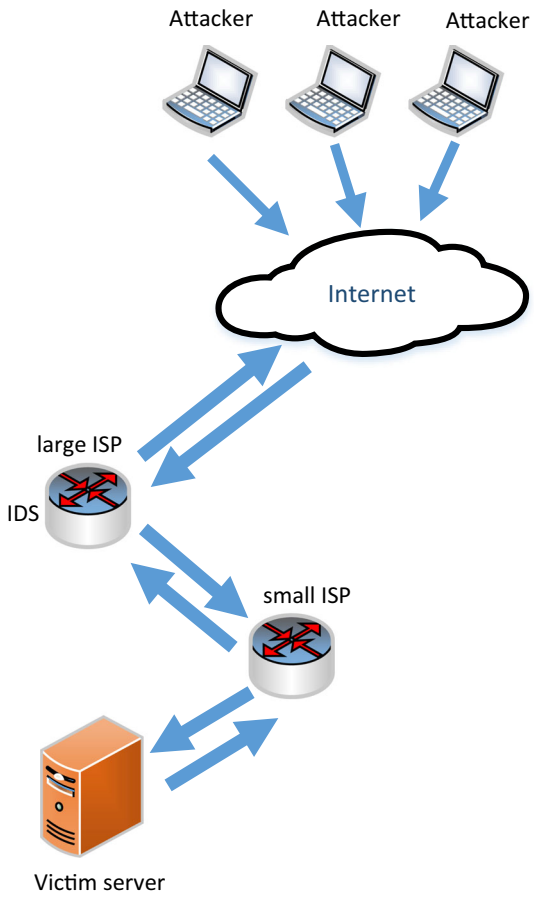

Fig. 1 Position of the IDS system

1.1 Background of TCP/IP protocol

TCP is a core protocol of the Internet protocol suite. TCP provides reliable, ordered, and error-checked delivery between applications communicating over the Internet. TCP enables reliability by using a three-way handshake to establish communication between two hosts and the four-way finish to terminate a connection. In the three-way handshake, communication is established in three distinctive segments. A client sends a control SYN packet to a server port to perform a request for communication. The server port must be in the listening state to be able to receive the packet. This packet is referred to as a request for communication. After receiving the control SYN packet, the server reserve connection resources to track the TCP state. After that, it responds to the client with the control SYN/ACK packet. This packet is referred to as a confirmation request. The client receives the control SYN/ACK packet and sends a control ACK packet, confirming it received server acknowledge. This packet is referred to as confirmation of the established connection.

After a successful three-way handshake, a connection is established and the data packets can be transferred. The data packets can be sent in both directions, from the client to the server as a request or, from the server to the client as a reply. For every data packet that the host receives, a control ACK packet is used to confirm delivery. The data ACK packet will be referred to as a DACK or data acknowledgement packet to distinguish a data control ACK packet from a control ACK packet.

When the connection is established, a four-way finish is performed in similar steps. The client initiates closing of the connection by sending a control FIN packet (graceful shutdown), or under certain conditions, a control RST packet which immediately resets the connection. The control FIN packet can also be initiated by the server. Upon receiving a control FIN packet, the server responds with a control ACK packet which acknowledges the connection closure. The server sends the control FIN packet to finish the connection. The client acknowledges the received control FIN packet by sending the control ACK packet. Once this packet has been received, the connection will be closed. In ordinary TCP behaviour, the number of control packets remains the same since every connection has its own closure.

1.2 Algorithms based on Counting Bloom Filter

DDoS detection should be able to analyse colossal TCP traffic. Space efficient data structures, known as Bloom filters are mostly used to efficiently store routing tables on networked devices. Most schemes use the algorithms based on the Counting Bloom Filter (CBF) to analyse large numbers of packets in real time [8,13]. The CBF is a structure based on the Bloom filter which inserts and deletes records. It recognizes a matching pair of connection packets by simplifying real input with hash functions. The matching pair is a control ACK packet correlated to the control SYN/ACK packet which was received earlier. This mechanism enables processing of a large amount of data over the input link, with minimal processing power and memory requirements.

Due to simplifying the input, CBF introduces recognition errors on matching pairs known as a false rate. The false rate is a data acknowledgement packet or another packet erroneously recognized as a control ACK packet. This issue is present within all algorithms that are based on the Bloom filter. The false rate can be reduced by tweaking the Bloom filter parameters. These parameters include the size of the field, the number of hash functions and a maximum number of elements used in the field. Enabling false rate reduction requires increasing the first two parameters. An increase of the number of hash functions and of field size requires additional memory and processing power.

In router data structure schemes, the false rate affects recognition of DDoS attacks. Our objective is to reduce the false rate by introducing a new structure called the Dual Counting Bloom Filter (DCBF). This new structure is proposed to be implemented over DDoS detection algorithm $SACK^2$ [14]. In [8], $SACK^2$ is presented as an algorithm that provides advantages for embedded systems implementation. Furthermore, the false rate is thoroughly analysed in [14]. These analyses are used to validate DCBF implementation.

The paper is organised as follows. Section 2 discusses related work on detection algorithms. Section 3 follows with the description of the CBF data structure and its false rate probability as a basis for improvement in proposed DCBF. In Sect. 4, a new structure to reduce a false rate error of matching pairs is proposed. Here, we will introduce an additional Bloom filter in the CBF data structure to improve the reliability of matching pair detection. The expected theoretical false rate in CBF and DCBF data structures is presented in Sect. 5. Using real Internet traces of a wide area network link, we will study the behaviour of the proposed DCBF. Finally, the conclusion is that false rate reduction is significant, providing minimal memory overhead and with minimal processing requirements, thus making it suitable for an embedded system.

2 Related work

Existing algorithms are classified based on router real-time traffic analysis capability, statistical algorithms and algorithms based on artificial intelligence [8]. Algorithms based on artificial intelligence require more development effort and additional resources. One example is the implementation of fuzzy logic in embedded systems which requires a huge number of real-time floating point calculations. Such demanding computations require the use of an additional Digital Signal Processor (DSP) [15,16]. Statistical algorithms require more memory for calculations and are difficult to implement in real time. In addition, statistical analysis limits the performance of network communication because of the overhead required for sampling packets in real time [17,18]. Traffic can be analysed in real-time on ISP routers which are implemented as embedded systems. There are multiple algorithms which are designed to reduce required resources and provide better detection [19].

It is challenging to analyse real-time traffic in time slots and make a decision for all traffic. Algorithms are based on the fact that TCP uses three segments to establish and four segments to terminate a connection. The expectation is that the number of control connection packets and the number of connection termination control packets are proportional in normal network traffic. If there is a significant disruption to this ratio, the existence of an attack can be proven.

The main drawback of these algorithms is that they rely on an assumption for choosing which control packet to use and how to define the correct ratio to prove the existence of the attack. There is a distinct difference in the behaviour of application protocols. The ratio threshold should be flexible enough to cover these differences and, at the same time, small enough to detect a small footprint attack. Given that the packet ratio in any network varies through time, it needs to be estimated in time intervals and subsequently used as a reference point in the following measurement cycle. Since there are three different control packets in the process of establishing a connection and four in terminating a connection, different packet ratios are available for examination. Additionally, some control packets are similar to others, hence distinguishing them becomes difficult [20].

A network router is used as the source of control packet information. The available data consists of the source and destination network addresses and the source and destination ports. In case of TCP protocol, sequence and acknowledgement numbers as additional data which describe the connection flow are used. Detection algorithms process this data to determine if an attack is occurring.

Detection errors are divided into two separate issues: false positives and false negatives. A false positive error is a detection of a non-existent ongoing attack, whereas a false negative error is a non-detected attack. Attackers hide attacks by cre-

ating a fraudulent joint packet and steer the algorithm to a misleading direction. A detected false positive error is caused by purposefully misleading recognition of the matching pair. The recognition error varies depending on the used control packet.

In the TCP protocol, specific types of packets, such as SYN, SYN/ACK, ACK, FIN, RST, are used to establish and close a connection. Several distinctive algorithms based on different monitored ratios of specific packet types have been developed.

Sun et al. [3,21] proposed a more robust scheme to detect SYN flooding attacks using the CBF which recognizes the corresponding FIN RST packet for the SYN packet. However, the SYN-FIN method cannot detect attacks mixed with spoofed FIN RST packets.

Kompella et al. [22] introduced a modified CBF called a Partial Completion Filter (PCF) to check the difference between SYN and FIN packets. The problem remains as a spoofed FIN RST packet can obstruct the algorithm.

Chen and Yeung [23,24] proposed using SYN-ACK pairs with the CBF. They did not reveal the method for differentiating between ACK and DACK. Even if a successful method is used, spoofing SYN and ACK packets may still obstruct the algorithm.

Nashat et al. [25] proposed a scheme based on the SYN-SYN/ACK protocol pair and packet header information inspection. The CBF is used to avoid the effect of SYN/ACK retransmission, and the Change Point Detection method is applied to avoid the dependency of detection on sites and access patterns. In case of DDoS, detecting a single malicious host in a large local network is difficult.

Ling et al. [26] proposed a defense mechanism that makes use of edge routers that connect end hosts to the Internet and detect and store whether the outgoing SYN, ACK or incoming SYN/ACK segment is valid. This is accomplished by maintaining a mapping table of the outgoing SYN segments and incoming SYN/ACK segments, and establishing the destination and source IP address database. A hashing function is used to establish mapping. This solution exploits ideas similar to the Bloom filter where higher filter levels are attenuated with respect to an earlier filter level characterised with a lossy distributed index that may introduce a false rate.

Sun et al. [14,27] proposed a more accurate and fast SYN flood detection method, named $SACK^2$. $SACK^2$ exploits the behaviour of SYN/ACK-ACK pairs. The CBF is used to recognize CliACK packet.

Halagan et al. [28] proposed, implemented and evaluated a new method for detection and type identification of SYN flood (DDoS) attacks. The method allows distinguishing the type of detected SYN flood attacks based on the Counting Bloom Filter. Their method uses a SYN and ACK pair. This solution is implemented in the network monitoring tool called KaTaLyzer.

The discussed methods are classified according to a matching pair which is then examined as four different types: SYN-FIN (RST), SYN-SYN/ACK, SYN-ACK and SYN/ACK-ACK.

The drawback of the SYN-FIN (RST) algorithm lies beyond the possibility of creating false FIN (RST) packets. It is impossible to determine the exact FIN packet for adequate SYN without a description of the entire connection. The moment FIN packets occur cannot be determined in advance since the connection duration is unpredictable. The application layer protocol can maintain a connection longer than the detection interval and thus induce a false positive error.

The SYN-SYN/ACK algorithm requires a significant difference in the number of packets. In a DDoS attack, local significance is minor, and it is impossible to be detected.

The SYN-ACK algorithm can be easily tricked. An attacker can send a mix of false SYN and ACK packets of random numbers to avoid detection.

In the SYN/ACK-ACK algorithm, false packets cannot be generated as correlated sequence numbers between pairs of packets exist. This method is recommended for detecting attacks on the Internet service provider routers. The disadvantage of the algorithm lies in the difficulty of distinguishing ACK from DACK, given that the ACK packet header contains the same flags as the DACK packet header. The difference can be observed by monitoring the packets' sequential numbers. The ACK packet is a response to SYN/ACK, and the DACK packet is the response to a packet of data after the connection is established. To recognize the difference, a record of header data has to be maintained. The easiest way to recognize DACK from ACK is to use the CBF as it provides detection abilities with minimal resources.

The disadvantage of the CBF comes from hashing functions, which simplify the input of a 160-bit packet header while computing the k hash function for it and producing hash values ranging from 1 to m. The number of values and proposed range directly influences the error rate of the structure. For greater ranges and values, the error rate becomes smaller, but the structure uses more resources (memory and processing power). It becomes useful to restrain the increase in the range and number of values in order to make this algorithm suitable for embedded system routers implementation [14].

Recent works [29,30] have proposed a new communication paradigm, namely, Information-Centric Networking (ICN) to solve several Internet problems like security, mobility, scalability and quality of service. Research on ICN has already provided notable solutions in many areas of Internet usage. Multiple algorithms based on the Bloom filter structure exist for improving source routing: the Free Riding Multicast (FRM), Line speed Publish/Subscribe Inter-Networking protocol (LIPSIN) and Bloomcast [29]. The Bloom filter is used to encode the source routing path from

the publisher to the subscriber. Using the Bloom filter introduces false positive errors due to two possible reasons: extra bandwidth and potential forwarding loops. False positives can be circumvented by adjusting the size and capacity of the Bloom filter and the number of hash functions. These parameters are used to define the fill factor of the Bloom filter structure [30].

3 Counting Bloom Filter data structure

The Bloom filter is a widely used space-efficient data structure that does not require a large amount of memory but does enable the identification of affiliation elements within a set. It consists of a data field size of m bits and k independently dispersed hash functions. A building block of a CBF structure is a Bloom filter that enables deletion by implementing each entry as a small counter [31]. Bloom filters are extensively used within the network security domain. In this section, we will discuss the main benefits and issues relating to its use in the $SACK^2$ algorithm.

3.1 Hash function analysis

The most commonly used hashing functions in embedded systems for linear transformations are a class of hashing functions called H_3 [32]. For an input array of one byte:

$$byte_i = <b_{i1}, b_{i2}, \ldots, b_{i8}>, \tag{1}$$

and the hash function H_3 is defined as:

$$h_i^l = d_{i1}^l b_1^i \oplus d_{i2}^l b_2^i \oplus d_{i3}^l b_3^i \oplus d_{i4}^l b_4^i \oplus \cdots \oplus d_{i8}^l b_8^i, \tag{2}$$

where d_{ij}^l are randomly generated numbers ranging from 1 to m. All other hashed functions are obtained using the following expression:

$$H_i^l = H_{i-1}^l \oplus h_i^l, \quad \forall l \in [1 \ldots k], \quad H_0^l = 0. \tag{3}$$

An input value is divided into bytes. For every bit within an input byte (i), a random value $d_{i(1-8)}$ is multiplied by the bit value $(b_{(1-8)}^i)$. On the resulting values, the exclusive disjunction is applied to form subscore (h_i) as defined in (2). The hash function is calculated as the exclusive disjunction between all byte subscores according to (3).

Generally, using the CBF to detect a matching pair of network packet requires hashing of 6-$tuple$ called $P_{syn/ack}$. This 6-$tuple$ input structure consists of: SIP as an IP address source, DIP as a destination IP address, SP is a source port, DP is a destination port, SEQ is a sequential packet number and $ASEQ$ is an acknowledge sequential packet number. This data is obtained from a SYN/ACK packet header:

$$P_{syn/ack} = <SIP, DIP, SP, DP, SEQ, ASEQ>. \tag{4}$$

Detecting a matching pair of SYN/ACK and ACK packets starts with the detection of the ACK packet. For every detected ACK or DACK packet which has the same flags in the packet header, and according to the matching flags in the packet header, a new 6-$tuple$ named P_{ack} is generated. P_{ack} tuple consists of the same members stored in different order:

$$P_{ack} = <DIP, SIP, DP, SP, ASEQ - 1, SEQ>. \tag{5}$$

The input structures $P_{syn/ack}$ and $P_{(d)ack}$ heading to the filter has a length of 20 bytes given that the IP address require 4 bytes, the source port 2 bytes and an accumulated sequence number of 4 bytes of memory. By using (2), the H_3 function is defined as:

$$H^l = d_1^l b_1 \oplus d_2^l b_2 \oplus d_3^l b_3 \oplus \cdots \oplus d_{160}^l b_{160}, \quad \forall l \in [1 \ldots k], \tag{6}$$

where d_{bit}^l is a randomly generated number ranging from 1 to m.

Since these functions exploit basic logic elements, they are suitable for efficient implementation in embedded systems [33]. Another possibility is implementation of hash functions in hardware via FPGA, which significantly reduces consumption of processing power [34].

In our case, for every $P_{syn/ack}$ and $P_{(d)ack}$ packet, a set of values is generated by applying a set of k hash functions:

$$R_{syn/ack}^l = H^l(P_{syn/ack}), \quad \forall l \in [1 \ldots k], \tag{7}$$

$$R_{(d)ack}^l = H^l(P_{(d)ack}), \quad \forall l \in [1 \ldots k]. \tag{8}$$

The result $R_{syn/ack}$ or $R_{(d)ack}$ of hash functions is a number ranging from 1 to m. For every SYN/ACK packet, the k values in the field 1 to m will be increased at the position $R_{syn/ack}^l$, $\forall l \in [1 \ldots k]$. If the value of the field overflows, the increase will be omitted. For every ACK packet, the k values in the field 1 to m will decrease if possible at position $R_{(d)ack}^l$, $\forall l \in [1 \ldots k]$. It is possible to decrease the fields if all the values in the required range are greater than zero. At the beginning, all values of the m fields are zero.

The purpose of the CBF data structure is to recognize the corresponding ACK packet of the previously received SYN/ACK packet. As the ACK packet has identical header flags as DACK, it is possible to recognize it as an ACK packet. In the case when the packet is propagated in the same direction and identical handshake in 6-tuple, only the sequential numbers are distinctive. For other connections types, all of the 6-tuple elements could be different. In normal network behaviour, every SYN/ACK will be matched with proper ACK. SYN/ACK will increase k values in the field m and

matched ACK will decrease the values in identical positions. Under certain conditions, there is a possibility that k values of the DACK from another or the same connection will be eventually decreased in field m. In this case, the assumption is that an appropriate ACK has arrived, and is no longer expected. When a matching ACK packet eventually arrives, the corresponding values in the field will already be empty, and this genuine matching ACK packet is erroneously recognized as a DACK and is disregarded. This situation is referred to as a false rate in the CBF data structure [35].

3.2 False rate probability

A false rate in the Bloom filter occurs due to the hashing which fails to guarantee unique mapping between an element and a group of indexes. The probability of a false rate for n elements of the input string in the CBF data structure with k independent functions on the field size m is determined using the following equation:

$$p_{fp} = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1 - e^{\frac{-kn}{m}}\right)^{k}. \tag{9}$$

The equation takes into account the ideal number of dispersed functions according to the following equation:

$$k = ln(2) \times \frac{m}{n}. \tag{10}$$

For the Bloom filter, this provides the best ratio of memory capacity and error probability. In this case, the error is:

$$p_{fp} \approx \left(\frac{1}{2}\right)^{k} \approx 0.6185^{\frac{m}{n}}. \tag{11}$$

For the case of the CBF, there is a possibility of an error that manifests in a lack of recognition of elements belonging to the input data set (i.e. false rate). This is possible due to the increasing and decreasing of field elements value. The occurrence of this error depends on the distribution of increasing and decreasing elements, as well as the number of elements in the array. Finding a theoretical solution to error propagation caused by the randomised nature of the assigned values is a real challenge.

According to Manna et al. [8], this technique generally gives rise to a large false positive rate caused by the Bloom filter data structure during network flow congestion. This false positive can affect the final decision. There are various solutions for improving false positives in the light of the rich Bloom filter data structure parameters. A recent work [36] has proposed that at each element insertion, hashed counters be incremented by a hashed variable increment, instead of a simple unit increment. Querying an element requires checking a particular counter value. In such cases, evaluation is based on purely randomised data and indicates a decrease in false positives by 53% while utilising the same amount of memory. Furthermore, another proposal to better eliminate false positives relies on Bloom-filter-based forwarding where Särelä et al. [37] proposed permuting the Bloom filter at each router. The permutation is accomplished by shuffling filter bits. This has been shown to sufficiently protect against loops and flow duplication of the path across which the packet has already traversed [7].

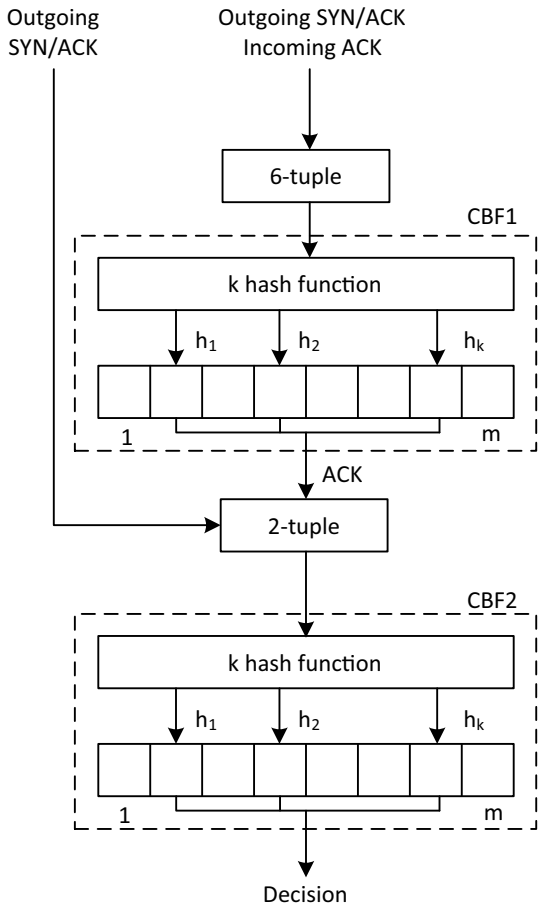

Fig. 2 Architecture of $SACK^2$ algorithm [14]

3.3 $SACK^2$ algorithm

The Bloom filter is a key structure in the $SACK^2$ SYN flood detection method, which exploits the behaviour of SYN/ACK-ACK pairs [14].

Figure 2 shows the architecture system for $SACK^2$. The CBF data structure is used twice: firstly, for recognizing the corresponding ACK packet for every SYN/ACK packet-CBF1, and secondly, for deciding on whether an attack exists-CBF2. If the CBF1 reaches a decision on the cor-

responding ACK packet, it will transfer a 2-*tuple* (the destination IP address and the destination port number) to CBF2.

CBF2 independently uses its own field and hash functions, which do not have to be of the same length as the field in CBF1. The primary input parameter to CBF2 is the SYN/ACK packet. For every SYN/ACK packet, a 2-*tuple* (source IP address and source port number) is hashed using hash functions and stowed into the CBF2 structure. This increases the counter values in CBF2. This 2-*tuple* structure is hashed using hash functions from CBF2. The results are used to decrease the values of the calculated positions in the field m of the CBF2.

If any counter in CBF2 exceeds the designated threshold, an attack is reported. The threshold is usually foreordained according to normal network traffic. The $SACK^2$ algorithm detects attacks in fixed time periods. In the case when returning values are below the threshold, the report of an existing attack is withdrawn. At the end of the period, the values of field m in CBF2 are reset to zero.

According to existing $SACK^2$ studies, a measured false rate supports the theory behind Bloom filters. Depending on the network traffic, the measured experimental false rate of CBF1 in a real environment may exceed 20% [14,27,35]. These peaks represent a drawback of the $SACK^2$ algorithm. CBF1 introduces a false rate for the matching pair SYN/ACK-ACK. This false rate is propagated into CBF2 which makes the decision about the existence of the DDoS attack.

In the following section, we will introduce a novel DCBF data structure which helps decrease false detection of matching packets and is compatible with $SACK^2$ algorithm.

4 Dual Counting Bloom Filter data structure

In order to reduce the false rate a new data structure called the Dual Counting Bloom Filter (DCBF) is introduced. This structure consists of an additional Bloom filter in CBF1 architecture. The responsibility of CBF1 is to analyse inverted input data. This additional CBF1 will be referred to as $\overline{CBF1}$. It is used for supplementary verification if a received ACK packet is part of the matching pair. This additional verification reduces the false rate. Figure 3 illustrates an improved architecture of $SACK^2$ with integrated DCBF as a replacement for the CBF1 data structure.

Input to the DCBF structure is the same as for CBF1 in the $SACK^2$ algorithm. This input is a 6-*tuple* called $P_{syn/ack}$. Information for this input is extracted from the packet header of the SYN/ACK which in turn is recognized by matching flags in the packet header:

$$P_{syn/ack} = <SIP, DIP, SP, DP, SEQ, ASEQ>. \quad (12)$$

Inversion for the $\overline{CBF1}$ is performed by complementing every bit in the $P_{syn/ack}$ input data. For every $P_{syn/ack}$ an inverted 6-*tuple* $\overline{P}_{syn/ack}$ is determined:

$$\overline{P}_{syn/ack} = <\overline{SIP}, \overline{DIP}, \overline{SP}, \overline{DP}, \overline{SEQ}, \overline{ASEQ}>. \quad (13)$$

Using (1), (2) and (3) hashed values are calculated for k hashed functions:

$$R^l_{syn/ack} = H^l(P_{syn/ack}), \quad \forall l \in [1 \ldots k]. \quad (14)$$

Simultaneously, $\overline{CBF1}$ in the DCBF data structure for every $\overline{P}_{syn/ack}$ generates a hashed value using the same set of k hash functions:

$$\overline{R}^l_{syn/ack} = H^l(\overline{P}_{syn/ack}), \quad \forall l \in [1 \ldots k]. \quad (15)$$

In (14) and (15), the same generated hash functions are used. Input data length is 20 bytes. The hash function is calculated according to (2) and (3):

$$H^l = d^l_1 \overline{b}_1 \oplus d^l_2 \overline{b}_2 \oplus d^l_3 \overline{b}_3 \oplus \cdots \oplus d^l_{160} \overline{b}_{160}, \quad \forall l \in [1 \ldots k]. \quad (16)$$

The resulting $R_{syn/ack}$ of hash functions is a number ranging from 1 to m. For every result obtained from a SYN/ACK packet, the k values in the field 1 to m will increase at position $R^l_{syn/ack}, \quad \forall l \in [1 \ldots k]$ in CBF1 within the DCBF data structure. If the value of the field overflows, the values will not increase.

The result $\overline{R}_{syn/ack}$ of the same hash functions is a number in a range 1 to m. For every result obtained from SYN/ACK packet, the k values in the field 1 to m increase at position $\overline{R}^l_{syn/ack}, \quad \forall l \in [1 \ldots k]$ in the $\overline{CBF1}$ in DCBF data structure. If the value of the field overflows, the values do not increase.

For each hashed result obtained from an ACK or DACK packet, the k values in the field 1 to m will decrease. This change will occur at position $R^l_{(d)ack}, \quad \forall l \in [1 \ldots k]$ from the CBF1 in the DCBF data structure. It is possible to decrease fields when all affected values are greater than zero. For the case when decreasing every hash function is not possible due to zero values, the conclusion is that this ACK packet has no matching SYN/ACK packet. $\overline{CBF1}$ in the DCBF data structure will not be changed or verified.

For the case when CBF1 determines an ACK packet, then the $P_{(d)ack}$ 6-*tuple* will be inverted:

$$\overline{P}_{ack} = <\overline{DIP}, \overline{SIP}, \overline{DP}, \overline{SP}, \overline{ASEQ - 1}, \overline{SEQ}>. \quad (17)$$

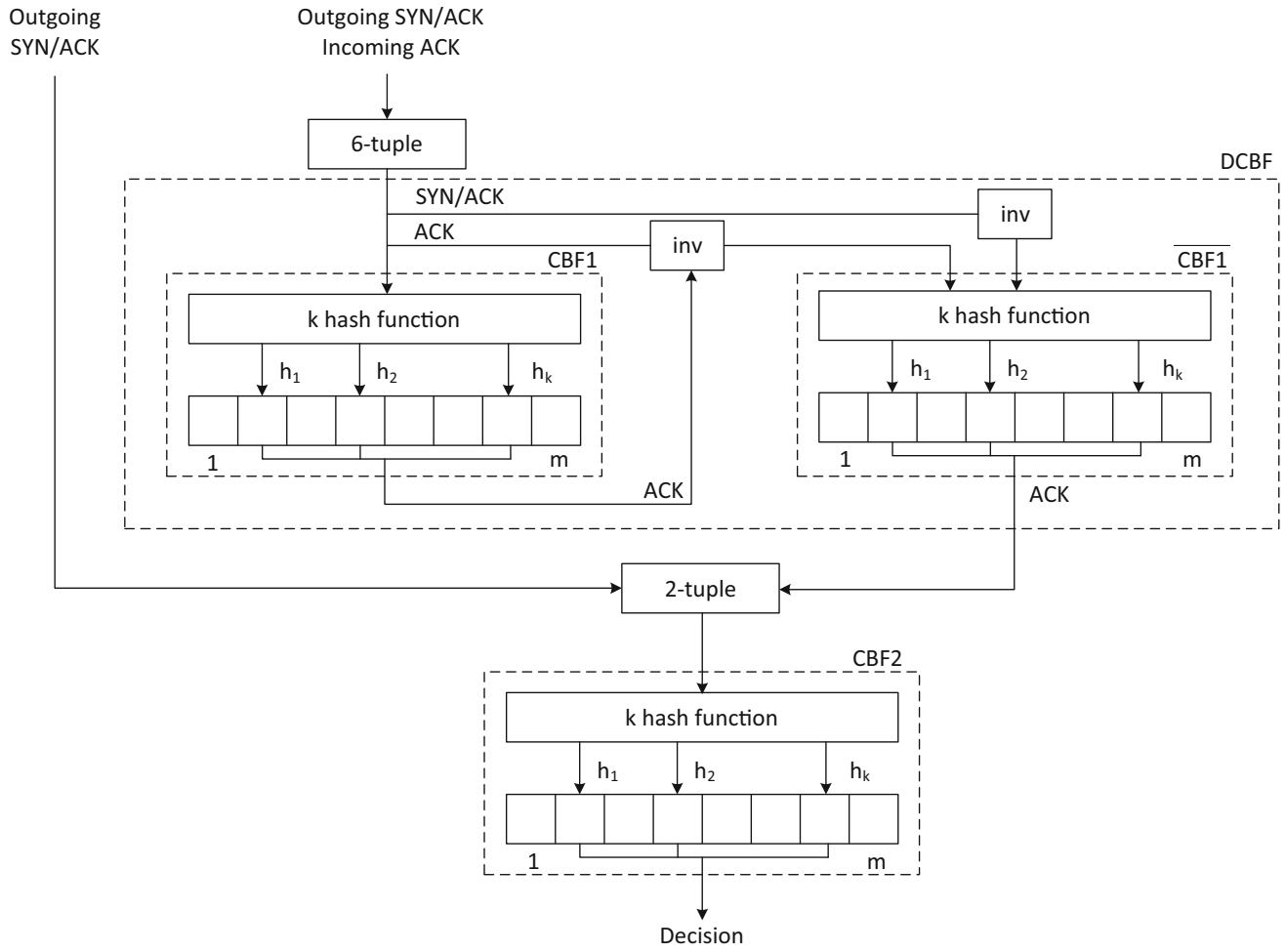

Fig. 3 DCBF improvement of $SACK^2$ algorithm

Hash values are calculated for (D)ACK according to (16):

$$\overline{R}^{l}_{(d)ack} = H^{l}(\overline{P}_{(d)ack}), \quad \forall l \in [1 \ldots k]. \tag{18}$$

This result is tested on $\overline{CBF1}$ in DCBF data structure to reduce the false rate for match verification. Every k values in the field 1 to m will be decreased if possible at the position $\overline{R}^{l}_{(d)ack}, \quad \forall l \in [1 \ldots k]$ from the $\overline{CBF1}$ in DCBF data structure. It is possible to decrease the fields if all the values that should be decreased are greater than zero.

In the case when the decrease of every hash function of the $\overline{CBF1}$ is not eligible, due to zero value, it is concluded that this ACK packet has no matching SYN/ACK packet.

Otherwise, if the decrease succeeds, the DCBF makes the decision that this ACK packet corresponds to the previous SYN/ACK packet. This decision is transferred to the CBF2 filter.

Obtained results from DCBF could be further processed by CBF2 using $SACK^2$ algorithm. Threshold determination and DDoS detection are implemented as described in [14].

5 DCBF evaluation

This section summarises the evaluation of results from the proposed structure DCBF and compares it with the used CBF structure in the $SACK^2$ algorithm. We present a theoretical approach to false rate estimation and resource utilisation. The experimental setup and results using real traffic measurements are discussed.

5.1 Theoretical evaluation

As discussed in Sect. 3, in the CBF the optimal value of the number of dispersed functions k, for a given size of the field m and the number of packets n, in order to minimise the false rate is given by (10). The minimum false rate value is therefore given by (11).

In the proposed DCBF, false rate should be reduced as the additional filter $\overline{CBF1}$ further verifies the input. The probability of false rate for each filter is estimated in line with the CBF equation. Given that the second Bloom filter has its

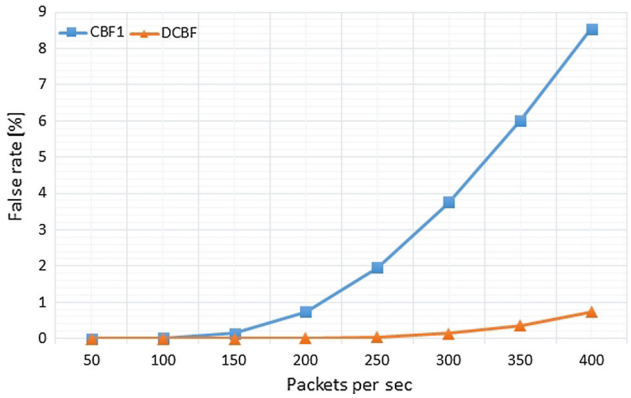

Fig. 4 Expected theoretical false rate in CBF1 and DCBF

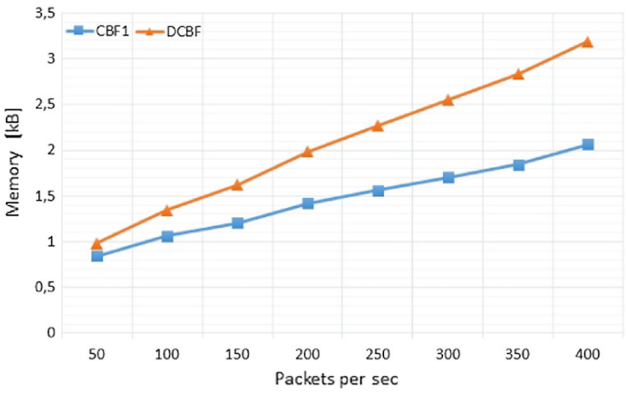

Fig. 5 Memory requirements of CBF1 and DCBF

own independent field m, a false positive for the DCBF is calculated as the false rate in the CBF data structure [35].

$$p_{fp} \approx \left(\frac{1}{2}\right)^k \times \left(\frac{1}{2}\right)^k \approx \left(\frac{1}{2}\right)^{2 \times ln(2) \times \frac{m}{n}} \approx 0.38255^{\frac{m}{n}}. \tag{19}$$

Figure 4 shows the expected theoretical ideal false rate values based on the number of packets given as an input to the filter. For a packet number greater than 200, the DCBF significantly decreases the false rate value and consequently improves the $SACK^2$ algorithm [8].

For a DCBF implementation in an embedded system, memory requirements should be considered. In the case of CBF1, the number of bits for each predetermined random number is $[log_2 m]$. The inputs to the first Bloom filter are packets of size $L = L_{syn} = L_{ack} = 160$ bits. We limited the theoretical analysis to the $n = 350$ as optimal, to take into account the physical limit of the maximum number of packets on the wide area network link in our experiment with $n = 400$ as a peak value. The most common used number of hash functions is $k = 4$, since it results in an average false rate of 5% as obtained by (11). Using the Eq. (10) the size of the field m is calculated using a known k value. To simplify the algorithm, the size of the field m is rounded up to the nearest power of two. For $m = 2048$ the resulting value is $n = 350$ and $k = 4$ (10). The analysis from Fan et al. [38] reveals that four bits per element in the field m should suffice for most network applications in order to avoid a counter overflow of the CBF data structure. In this case, the probability of overflow is $1.37 \times m \times 10^{-15}$. Memory usage for CBF1 is:

$$M_{CBF1} = 4 \times m + k \times L \times log_2 m. \tag{20}$$

In our case and based on the chosen parameters, the resulting requirement is 1.84 kB for the CBF1 Bloom filter. The memory content can be divided into two parts. The first part of the required memory is used for storing the CBF field, and the second part for storing random numbers used by dispersed functions. The DCBF utilising an additional Bloom filter requires additional storage for the m field. Randomly generated numbers (d) in the hash functions are the same for both Bloom filters. The only difference is in input data which is inverted for the second Bloom filter. The total memory requirement of the DCBF is:

$$M_{DCBF} = 2 \times 4 \times m + k \times L \times log_2 m. \tag{21}$$

For this case, the memory requirement is 2.83 kB for the DCBF Bloom filter. Thus, in a worst-case scenario for our measurement link, the memory requirement is increased by 0.98 kB. According to Sun et al., implementing the $SACK^2$ algorithm requires 18 kB of memory for CBF1 and CBF2 [27]. CBF2 uses an additional 32 bits per element of the field m. CBF1 in $SACK^2$ uses 1.84 kB. As a result, by replacing CBF1 with DCBF in $SACK^2$, the required additional memory increases by 5.56%.

Figure 5 shows the memory consumption required by CBF1 and DCBF depending on the number of packets.

5.2 Experimental results

A study was conducted to verify the improvement of the false rate for a DCBF. CBF1, part of the $SACK^2$ algorithm and the proposed DCBF were simulated in line with the presented algorithms. The measurements for input data were acquired on a wide area network link. Traffic was aggregated over a 6-month period and stored in a database.

Traffic varied from $n = 6$ to a maximum possible $n = 400$ packets per second. Smaller traffic patterns when $n < 50$ were isolated because the false rate did not occur for the CBF1 or the DCBF. This was as expected and in line with (10) for $n = 50$ where $m = 256$ is sufficient, which is far smaller than the used $m = 2048$. Experimental evaluation with the number of packets exceeding $n = 50$ leads to a false rate

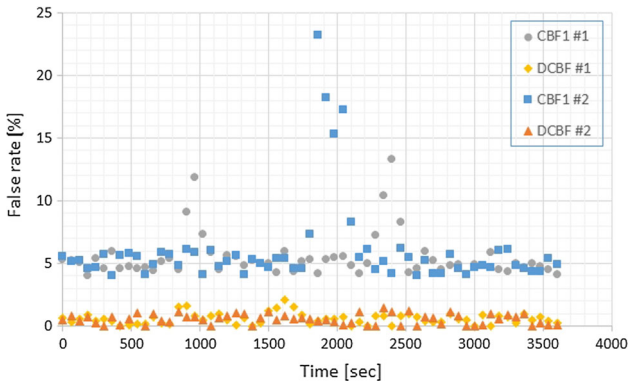

Fig. 6 False rate measurement of CBF1 and DCBF

Table 1 CBF1 and DCBF false rate

False rate	CBF1	DCBF
Minimal	4.03	0
Average	6.05	0.59
Maximum	23.25	2.11

starting to appear. Isolation of smaller traffic patterns results in an effective average traffic of $n = 200$.

The recorded traffic was analysed and modified to provide ideal matching pairs. Such traffic modification ensures consistent traffic data without erroneous network conditions or possibly capturing an attack. In this way, every attack detection in our analysis can be interpreted as a pure false rate. This traffic data was analysed using CBF1 and the DCBF algorithm. We captured a large number of samples, analysed and presented the most interesting samples, which confirmed the existence of the false positive rate.

In Fig. 6, we present two examples of traffic in a one-hour period, where the samples were taken every 60 s. The graph shows a false rate for each algorithm (CBF1, DCBF). The stream designated with the same number was obtained for the same input traffic.

Table 1 shows the minimal, average and maximum false rate values of the total measured traffic for each algorithm. Achieved average DCBF false rate improvement is 89,53%, minimal value is 72,26% and the maximal value is 100% with respect to CBF1.

When comparing these obtained results with Eqs. (11) and (19), a certain similarity can be observed. CBF1 and the DCBF data structure accumulate and delete values contrary to the Bloom filter. Additionally, CBF1 and DCBF reset the entire field m when any counter reaches half of its maximum. These differences cause deviations in experimental measurements when compared to what is expected theoretically.

In line with all these measurements, a DCBF false rate outperforms CBF1 on average by 90%. Additionally, DCBF in a worst case scenario, requires approximately an additional 1 kB of memory. The computational overhead of the proposed solution introduces 7–12% activation of the additional Bloom filter $\overline{CBF1}$ while at the same time significantly improving the false rate. Future work will verify the proposed algorithm improvements in comparison to the Bloom-filter-based forwarding algorithm.

6 Conclusion

This paper proposes a novel DCBF structure. The objective is to decrease the false rate with minimal additional resources. The proposed solution is compared to the router data structure scheme algorithm $SACK^2$. Router data structure is used to analyse the ratio of the connection and connection termination control packets in TCP communication. Every packet header needs to be examined to determine the ratio of the control packets.

Processing complete headers require a significant amount of memory, especially on a fast link or during network congestion. The data is organised into a memory efficient CBF data structure to reduce memory requirements. The algorithms based on the CBF require fewer resources (e.g. memory and processing power), making it suitable for deployment in an embedded environment.

The negative side of the Bloom filter is the error caused by false recognition of a matching SYN/ACK and ACK pair. The proposed DCBF decreases the false recognition rate by approximately of 90%. However, additionally exploiting the CBF increases resource requirements. The increase in memory consumption is the biggest drawback of the DCBF structure for implementation in an embedded system (e.g. a router). The proposed solution for the case of a flow of network packets of $n = 400$ required an additional 1.13 kB of memory compared to the standard CBF, which means it is still suitable for implementation in an embedded system.

Analyses showed that the additional 5.56% of memory used in the DCBF compared to the $SACK^2$ algorithm improves the accuracy of SYN/ACK-ACK pair detection by 90%. Moreover, DDoS attack detection using this algorithm improved immensely in the case of a minor attack.

In the future, the plan is to explore the possibility of applying the proposed solution to other algorithms based on the CBF or Bloom filters. Further verification will include data from different network locations and a comparison of different existing algorithms. Additionally, this paper assumes that a false rate error of 20% in the CBF1 data structure will be propagated to the error rate in CBF2. This is expected to affect detection of DDoS attacks. This correlation could be further examined. The verification of the new improved $SACK^2$ algorithm during the different rate of DDoS attack is a subject of future work.

Acknowledgements The authors would like to thank Central Informatics Support staff at Zagreb University of Applied Sciences for gathering the data.

Compliance with ethical standards

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Cisco. (2016). Annual Security Report 2016. http://www.cisco.com/c/dam/assets/offers/pdfs/cisco-asr-2016.pdf. Accessed Jan 2016.
2. Zhang, G., Fischer-Hübner, S., & Ehlert, S. (2010). Blocking attacks on SIP VoIP proxies caused by external processing. *Telecommunication Systems*, *45*(1), 61–76.
3. Sun, C., Fan, J., & Liu, B. (2007). A robust scheme to detect SYN flooding attacks. In *Second International Conference on Communications and Networking* (pp. 397–401).
4. Li, L., & Lee, G. (2005). DDoS attack detection and wavelets. *Telecommunication Systems*, *28*(3–4), 435–451.
5. Zlomislić, V., Fertalj, K., & Sruk, V. (2017). Denial of service attacks, defences and research challenges. *Cluster Computing The Journal of Networks, Software Tools and Applications*, *20*(1), 1–11.
6. DDoS Attacks in Q4 2015. Kaspersky Lab Report. https://securelist.com/analysis/quarterly-malware-reports/73414/kaspersky-ddos-intelligence-report-for-q4-2015/. Accessed Jan 2016.
7. Markku, A., Aura, T., & Särelä, M. (2014). Denial-of-service attacks in Bloom-filter-based forwarding. *IEEE/ACM Transactions on Networking (TON)*, *22*(5), 1463–1476.
8. Mehdi, M. A., & Amphawan, A. (2012). Review of syn-flooding attack detection mechanism. *International Journal of Distributed & Parallel Systems*, *3*(1), 99–117.
9. Scarfone, K., & Mell, P. (2010). *Guide to intrusion detection and prevention systems (IDPS)* (NIST SP 800-94). Washington, DC: Computer Security Resource Center, National Institute of Standards and Technology, U.S. Department of Commerce.
10. Wang, G., Xu, M., & Huan, X. (2012). Design and implementation of an embedded router with packet filtering. In *Proceedings—2012 IEEE Symposium on Electrical and Electronics Engineering, EEESYM 2012* (pp. 285–288).
11. Mittal, A., Shrivastava, A. K., & Manoria, M. (2011). A review of DDOS attack and its countermeasures in TCP based networks. *International Journal of Computer Science & Engineering Survey (IJCSES)*, *2*(4), 177–187.
12. Ma, X., & Chen, Y. (2014). DDoS detection method based on chaos analysis of network traffic entropy. *IEEE Communications Letters*, *18*(1), 114–117.
13. Broder, A., & Mitzenmacher, M. (2003). Network application of Bloom filters: A survey. *Internet Mathematics*, *1*(4), 485–509.
14. Sun, C., Hu, C., Tang, Yi, & Liu, B. (2009). More accurate and fast SYN flood detection. In *Proceedings of 18th International Conference on Computer Communications and Networks* (pp. 1–6).
15. Farkaz, F., & Halasz, S. (2006). Embedded fuzzy controller for industrial applications. *Acta Polytechnica Hungarica*, *3*(2), 41–63.
16. Xia, Z., Lu, S., Li, J., & Tang, J. (2010). Enhancing DDoS flood attack detection via intelligent fuzzy logic. *Informatica (Slovenia) An International Journal of Computing and Informatics*, *34*(4), 497–507.
17. Kawahara, R., Ishibashi, K., Mori, T., Kamiyama, N., Harada, S., & Asano, S. (2007). Detection accuracy of network anomalies using sampled flow statistics. In *Global Telecommunications Conference 2007, GLOBE-COM '07* (pp. 1959–1964). IEEE.
18. Kanwal, G., & Rshma, C. (2011). Detection of DDoS attack using data mining. *International Journal of Computing and Business Research (IJCBR)*, *2*(1), 1–10.
19. Prathibha, R. C., & Rejimol Robinson, R. R. (2014). A comparative study of defense mechanisms against SYN flooding attack. *International Journal of Computer Applications*, *98*(1), 16–21.
20. Fall, R. K., & Stevens, R. W. (2012). *TCP/IP illustrated, volume 1: The protocols*. Addison-Wesley Professional Computing Series. New York: Pearson Education.
21. Sun, C., Fan, J., Shi, L., & Liu, B. (2007). A novel router-based scheme to mitigate SYN flooding DDoS attacks. In *IEEE INFOCOM (Poster), Anchorage, Alaska, USA*
22. Kompella, R., Singh, S., & Varghese, G. (2007). On scalable attack detection in the network. *IEEE/ACM Transactions on Networking*, *15*(1), 14–25.
23. Chen, W., Yeung, D. Y. (2006). Defending against TCP SYN flooding attacks under different types of IP spoofing. In *International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL06)* (pp. 38–42).
24. Chen, W., & Yeung, D. Y. (2006). Throttling spoofed SYN flooding traffic at the source. *Telecommunication Systems*, *33*(1), 47–65.
25. Nashat, D., Juang, X., & Horiguchi, S. (2008). Router based detection for low-rate agents of DDoS attack. In *2008 International Conference on High Performance Switching and Routing* (pp. 177–182).
26. Ling, Y., Gu, Y., & Wei, G. (2009). Detect SYN flooding attack in edge routers. *International Journal of Security and its Applications*, *3*(1), 31–45.
27. Sun, C., Hu, C., & Liu, B. (2013). *SACK*^{2}: Effective SYN flood detection against skillful spoofs. *IET Information Security*, *6*(3), 149–156.
28. Halagan, T., Kovacik, T., Truchly, P., & Binder, A. (2015). Syn flood attack detection and type distinguishing mechanism based on Counting Bloom Filter. In *Information and Communication Technology: Third IFIP TC 5/8 International Conference, ICT-EurAsia 2015, and 9th IFIP WG 8.9 Working Conference, CONFENIS 2015, Held as Part of WCC 2015, Daejeon, Korea, 4–7 Oct 2015, Proceedings* (pp. 30–39). Springer, New York.
29. Alzahrani, A. B., Vassilakis, G. V., & Reed, J. M. (2014). Selecting Bloom-filter header lengths for secure information centric networking. In *2014 9th International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP)* (pp. 628–633). IEEE.
30. Alzahrani, B., Vassilakis, V., Alreshoodi, M., Alarfaj, F., & Alhindi, A. (2016). Proactive detection of DDOS attacks in Publish-Subscribe networks. *International Journal of Network Security & Its Applications (IJNSA)*, *8*(4), 1–15.
31. Blustein, J., & El-Maazawi, A. (2002). Bloom filters—A tutorial, analysis, and survey. Faculty of Computer Science, Dalhousie University. https://www.cs.dal.ca/sites/default/files/technical_reports/CS-2002-10.pdf. Accessed Jan 2016.
32. Ramakrishna, M. V., Fu, E., & Bahcekapili, E. (1997). Efficient hardware hashing functions for high performance computers. *IEEE Transactions on Computers*, *46*(12), 1378–1381.
33. Ramakrishna, M. V., Fu, E., & Bahcekapili, E. (1994). A performance study of hashing functions for hardware applications. In *Proceedings of International Conference on Computing and Information* (pp. 1621–1636).
34. Harwayne-Gidansky, J., Stefan, D., & Dalal, I. (2009). FPGA-based SoC for real-time network intrusion detection using Counting Bloom Filters. In *IEEE Southeastcon 2009* (pp. 452–458).
35. Tabataba, F.S., & Hashemi, M.R. (2011). Improving false positive in Bloom filter. In *2011 19th Iranian Conference on Electrical Engineering* (pp. 1–5).

36. Rottenstreich, O., Kanizo, Y., & Keslassy, I. (2014). The variable increment counting Bloom filter. *IEEE/ACM Transactions on Networking*, *22*(4), 1092–1105.
37. Särelä, M., Rothenberg, C. E., Aura, T., Zahemszky, A., Nikander, P., & Ott, J. (2011). Forwarding anomalies in Bloom filter-based multicast. In *INFOCOM, 2011 Proceedings IEEE* (pp. 2399–2407).
38. Fan, L., Cao, P., Almeida, J., & Broder, A. Z. (2000). Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking (TON)*, *8*(3), 281–293.

Ivica Dodig is pursuing a PhD in a Computer Science program at Faculty of Electronics and Computing in Zagreb, University of Zagreb, Croatia. He has a MS in Computer Sciences from Faculty of Electronics and Computing in Zagreb, Croatia. Main area of research is Denial of Service attacks. Currently working at Zagreb, University of Applied Sciences as a senior lecturer.

Vlado Sruk is an Associate Professor at the Department of Electronics, Microelectronics, Computer and Intelligent Systems at Faculty of Electrical Engineering and Computing. He obtained Ph.D. degree in computer science from the University of Zagreb in 1998. He participated as a researcher in international scientific projects. His current research interests are in the areas of multicore embedded systems, mobile computing, high-performance computing with emphasis on memory architectures, and state-of-the-art concepts and techniques in multicore software engineering and fault-tolerant computing. He is a member of IEEE and ACM society. He participates in conference international programs committees, and he serves as a technical reviewer for various international journals and conferences.

Davor Cafuta is a PhD candidate in a Computer Science program at Faculty of Electronics and Computing in Zagreb, University of Zagreb, Croatia. He has a MS in Computer Sciences from Faculty of Electronics and Computing in Zagreb, Croatia. Main area of research is network security and embedded systems. Currently working at Zagreb, University of Applied Sciences as a senior lecturer.