Computer Networks 57 (2013) 4047–4064

Contents lists available at ScienceDirect

Computer Networks

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

Survey Paper

Bloom filter applications in network security: A state-of-the-art survey

Shahabeddin Geravand [a,*], Mahmood Ahmadi [b]

[a] Department of Computer Engineering, Islamic Azad University of Arak, Arak, Iran
[b] Department of Computer Engineering, University of Razi, Kermanshah, Iran

ARTICLE INFO

Article history:
Received 13 September 2012
Received in revised form 14 May 2013
Accepted 6 September 2013
Available online 14 September 2013

Keywords:
Bloom filters
Security
Network processing

ABSTRACT

Undoubtedly, dealing with security issues is one of the most important and complex tasks various networks face today. A large number of security algorithms have been proposed to enhance security in various types of networks. Many of these solutions are either directly or indirectly based on *Bloom filter* (BF), a space- and time-efficient probabilistic data structure introduced by Burton Bloom in 1970. Obviously, Bloom filters and their variants are getting more and more consideration in network security area. This paper provides an up-to-date survey of the application of BFs and their variants to improve performance of the approaches proposed to address security problems with different types of networks.

1. Introduction

Security has always been a major concern for networked systems administrators and users. With the increasing use of high-speed networks and also increasing demand for specific technologies such as wireless, network security has become a complex challenge and a priority issue. Many approaches have been proposed to achieve the various security goals. In these approaches, a variety of techniques and data structures have been used to address the security concerns in an efficient manner. On the other hand, there are typically umpteen numbers of data items that need to be stored, queried and updated in the network environment. Therefore, the fact is concluded that space and time are two important factors that should be taken into consideration by the security approaches, especially in the specific networks, such as sensor networks, which suffer from severe limitations.

A probabilistic data structure that has been widely utilized in this field is *Bloom filter* (BF), which was introduced by Burton Bloom in 1970 [1]. BF is a simple, memory- and time-efficient randomized data structure for succinctly representing a set of elements and supporting set membership queries. These properties of BF make it very attractive to be utilized for many security applications. Initially, BF was applied to database applications, spell checkers and file operations [2–4]. In recent years, BFs and their variants have been widely used in networking applications, such as resource routing, security, and web caching [5,6].

This paper provides a state-of-the-art survey on the applications of BFs in the field of network security. A hierarchical classification of the various security-related schemes which are either directly or indirectly based on BFs is provided. In the first level of the classification, we classify networking environments into two categories: *wireless networks* and *wired networks*. This is because they are different from each other in some security aspects. In the second level, each category is broken up further into several subsections each of which explores a specific field of BF applications. Note that we only focus on the idea behind the approaches without discussing implementation details. It should be noted, however, that our goal of making this survey is not providing an exact classification of security attacks for different networks. But, we intend to review where BFs and their variants have been used to improve the efficiency of the different security schemes.

* Corresponding author. Tel.: +98 936 502 6483.
E-mail addresses: shgeravand@gmail.com (S. Geravand), m.ahmadi@razi.ac.ir (M. Ahmadi).

1389-1286/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2013.09.003

It is our hope that it provides useful information for them who want to investigate in this scope and use BFs in new applications.

The rest of this survey article organized as follows. Section 2 provides an introduction to the theory of standard BF. Moreover, this section briefly introduces the basic idea behind some important variants of BFs used in security-related schemes. Section 3 surveys the contribution of BFs to the network security according to the hierarchical classification of the existing schemes. Finally, Section 4 concludes the survey with a brief summary on the Bloom filters applications.

2. Bloom filters and theirs variants

A Bloom filter is a data structure which can store the elements of a set in a space-efficient manner, if a small error is allowed when testing for elements in the Bloom filter. In Section 2.1, these basic properties of Bloom filters are described. In the years after the introduction of the Bloom filter, several data structures based on the basic filter were presented by different researchers. These variants are described in Section 2.2.

2.1. Preliminaries of Bloom filters

Information representation and query processing are two core problems of many computer applications, and are often associated with each other. Representation means organizing information according to some formats and mechanisms, and making information operable by the corresponding method. Query processing means making a decision about whether an element with a given attribute value belongs to a given set. For this purpose, BF can be an optimal candidate.

A Bloom filter, conceived by Burton Howard Bloom in 1970, is a simple space-efficient randomized data structure for representing a set in order to support membership queries [1]. BFs may yield a small rate of false positives in membership queries; that is, an element might be incorrectly recognized as member of the set. Although Bloom filters allow false positives, for many applications the space savings and locating time constantly outweigh this drawback when the probability of false positive can be made sufficiently small.

Initially, BF was applied to database applications, spell checkers and file operations [2–4]. In recent years, BFs have received a great deal of attention in networking applications, such as peer-to-peer applications, resource routing, security, and web caching [5,6]. A survey on the applications of Bloom filters in distributed systems can be found in [7]. BFs are also being used in practice. For instance, Google Chrome uses a Bloom filter to represent a blacklist of dangerous URLs.

The idea of standard BF is to allocate vector A of m bits, initially all set to 0, for representing a set $S = \{x_1, x_2, \ldots, x_n\}$ of n elements. The BF uses k independent hash functions $h_1, h_2, \ldots, h_k$, each with range $\{0, \ldots, m-1\}$. A BF is constructed in two phases: programming phase and querying phase [1,5]. In the programming phase, each element $x \in S$ is hashed by k independent hash functions. Then, all the bits at positions $A[h_i(x)]$ in A are set to 1 for $(1 \leqslant i \leqslant k)$. Fig. 1 depicts the pseudocode for insertion of n elements.

A particular position in the vector might be set to 1 multiple times, but only the first time has an effect. In the querying phase, to query for an element y, we check the bits at positions $h_i(y)$. If any of the bits at these positions are 0, the element is definitely not in the set. Otherwise, either the element is in the set, or the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive. Fig. 2 depicts the pseudocode for querying an element. The more elements that are added to the set, the larger the probability of false positives.

The percentage of false positive of a Bloom filter can be minimized by tuning the three parameters: (i) number of elements (n) added to generate the Bloom filter. In most cases, this parameter is defined by the application and, thus, cannot be controlled. (ii) Number of bits used in a Bloom filter (m). m can be used in order to minimize false positives but obviously the larger the value of m the less compact representation. (iii) Number of hash functions (k) used to create the Bloom filter. The larger k the higher processing overhead (CPU usage) especially if hash functions perform complex operations.

Fig. 3 depicts the mentioned process. In Fig. 3, three elements x_1, x_2, and x_3 are separately hashed by 3 hash functions and then the corresponding bits in A are set to 1. To check if the element y_1 is in the set approximated by A, we check whether all $A[h_i(y_1)]$ are 1. As depicted in Fig. 3, because the bit position 8 is not 1, we surely conclude that y_1 is not a member of the set. Since all the three bit positions related to y_2 are set to 1, we conclude that y_2 is a member, although this may be wrong due to the false positive probability.

There is a trade-off between the probability of false positive and the length m of the BF array [1,5]. It has been proven that the probability of false positive (fp) is equal to:

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

Input: Set of n elements;
Output: Bloom filter A;
A = allocate m bits initialized to 0;
for *each* x_i *in the set* **do**
  for $j \leftarrow 1$ **to** k **do**
    $A[h_j(x_i)] \leftarrow 1$;

Fig. 1. Pseudocode for programming phase.

Input: An element y;
Output: True/False;
for $j \leftarrow 1$ **to** k **do**
  If $A[h_j(y)] = 0$ return False;
return True;

Fig. 2. Pseudocode for querying phase.

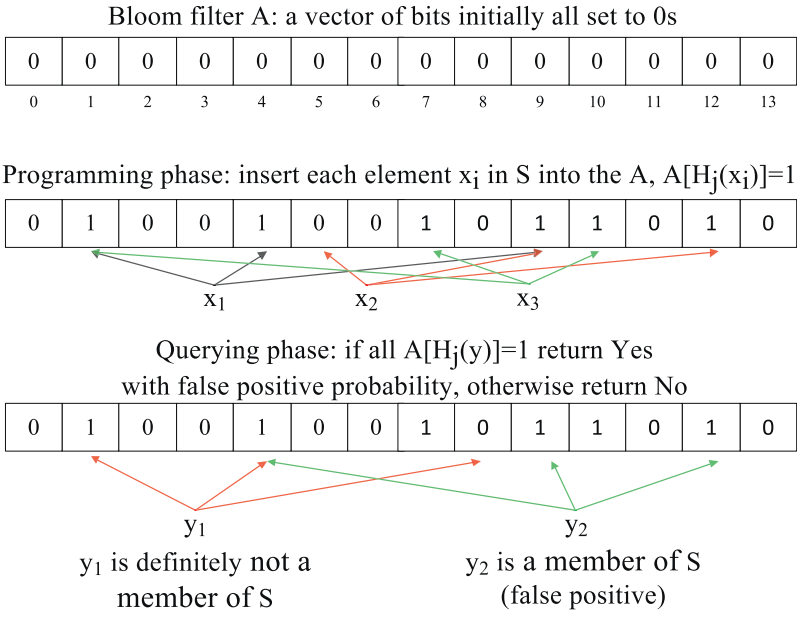

Fig. 3. Insert and query operations in standard Bloom filter.

Now it is clear that the optimal number of hash functions k, which minimize fp, can be easily found by taking the derivative of the above equation [1,5]. Therefore:

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

In the last decade, a number of extensions of the original BF have been developed by researchers to address its limitations. Subsequently, we briefly review some of the important variants utilized in the network security-related schemes discussed in this research.

2.2. Bloom filter variants

Several improvements have been proposed over the original Bloom filter. In this section, some well-known variants of Bloom filters are presented.

2.2.1. Counting Bloom filter

The standard BF can only allow for insert operation on a BF. We cannot remove an element from the BF because we might zero a bit that was also set by another element and so mark it as not being in the filter as well. To address this drawback, *counting Bloom filter* (CBF) was introduced by Fan et al. [6]. A counting Bloom filter replaces the array of bits with the array of counters. In fact, each position is a counter, allowing insert and delete operations on the CBF. Whenever an element is added to or deleted from the CBF, the corresponding counters are incremented or decremented, respectively. The size of the counter must be chosen large enough to avoid counter overflow. The analysis performed in [6] shows that 3 or 4 bits per counter works well for most applications.

2.2.2. d-Left counting Bloom filter

Bonomi et al. [8] presented a data structure based on d-left hashing and fingerprints that is functionally equivalent to a counting Bloom filter, but saves approximately a factor of two or more space. The d-left hashing scheme divides a hash table into d subtables that are of equal size. Each subtable has n/d buckets, where n is the total number of buckets. Each bucket has capacity for c cells, each cell being of some fixed bit size to store a fingerprint of the element along with a counter. When an element is placed into the table, following the d-left hashing technique, d candidate buckets are obtained by computing d independent hash values of the element. A hash-based fingerprint $f_x = H(x)$ is stored in the bucket that contains more empty cells (i.e., least inserted elements per bucket). In case of a tie, the element is placed in the bucket of the leftmost subtable with the smallest number of elements examined. Element lookups use parallel search of the d subtables to find the fingerprint and obtain the value of the counter. In case of a deletion the counter is decremented by one [8].

2.2.3. Compressed Bloom filter

Bloom filters can be used in distributed protocols where systems need to share information about what recourses they have, like URLs [6]. In such environments, Bloom filters are usually sent as messages over the network. In order to reduce the number of bits broadcast, the false positive probability, and/or the amount of computation per lookup, the idea of compressed Bloom filter was introduced by Mitzenmacher [9]. In addition to the three fundamental metrics for original Bloom filters (i.e., k, m and n), compressed Bloom filter introduces another factor, called the transmission size z, which refers to the size of the data

that needs to be sent over the network. According to [9], Bloom filters can be compressed to improve their performance by achieving either a lower false positive rate with the same memory size or a smaller memory size with the same false positive rate. Mitzenmacher shows in [9] that compressing Bloom filters might lead to significant bandwidth savings at the cost of higher memory requirements (larger uncompressed filters) and some additional computation time to compress the filter that is sent across the network.

2.2.4. Bloomier filter

Whereas the standard Bloom filter can only support membership queries on the elements of the set S, Bloomier filters are able to store the membership function $f = S \rightarrow [0, 1]$ [10]. Bloomier filters can encode arbitrary functions and allow us to associate values with a subset of the elements of the filter. Bloomier filters generalize Bloom filters to functions while maintaining their economical use of storage. Also, they allow for dynamic updates to the function, provided the support of the function remains unchanged. To build a meta-database, for example, the Bloomier filter keeps track of which database contains information about each bucket. Therefore, it allows us to directly access the relevant database.

2.2.5. Space-code Bloom filter

Per-flow traffic measurement is critical for usage accounting, traffic engineering, and anomaly detection. Kumar et al. [11] introduced a novel technique for measuring perflow traffic approximately, which is based on space-code Bloom filters. A space-code Bloom filter is an approximate representation of a multiset in order to answer queries of the form "Is an element x in a multiset?" and "How many occurrences of x are there in multiset?". SCBF uses several Bloom filters and maximum likelihood estimation in order to represent accurate estimates of element counts for all flows regardless of their sizes, at very high-speed. Each element in this multiset is a traffic flow and its multiplicity is the number of packets in the flow. The space-code Bloom filter is made up of l groups of hash functions, each group can be viewed as a traditional Bloom filter. To insert an element x into the filter, the element is hashed through one group of the hash functions chosen randomly, and then the corresponding bits, $A[h_1^i(x), h_2^i(x), \ldots, h_k^i(x)]$ in the filter are set to 1. To query for a flow y, first counts the number of groups that y has matched and then the result is used to estimate multiplicity of y in the multiset [11].

2.2.6. Dynamic Bloom filter

Standard Bloom filters are suitable for representing static sets whose size is known in advance and does not change over time. Dynamic Bloom filters [12] address this drawback by dynamically creating new filters as they are needed. The basic idea of dynamic Bloom filters is to represent a dynamic set A with a dynamic $s \times m$ bit matrix that consists of s standard Bloom filters. A dynamic Bloom filter initially consists of one active Bloom filter. That is, the initial value of s is one. The elements of the set are then inserted into this active filter. Before the false positive rate of the active filter starts growing fast, we simply switch to a new filter, store the old one and then add 1 to s. Only the last Bloom filter of a DBF is always active, others are inactive. To query for an element y, we try to find a standard Bloom filter with all bits $h_j(y)$ set to 1. If the result is false, the element is definitely not in the set. Otherwise, we believe that $y \in A$ with some false positive probability. Dynamic Bloom filter has been intended for a number of distributed environments, especially those in which new data is inserted (and potentially removed) frequently [12].

There are several more of these variants, such as distance-sensitive Bloom filter [13], spectral Bloom filter [14], generalized Bloom filter [15], scalable Bloom filter [16], split Bloom filter [17], attenuated Bloom filter [18], and incremental Bloom filter [19]. A brief description of all the mentioned variations can be found in [7].

3. Applications of Bloom filters in network security

In this section, we review the network security schemes which are directly or indirectly based on BFs and their new variants. We conduct a taxonomy of uses of the BFs in different networks as shown in Table 1. In the wireless networks category, application of BFs in various types of wireless networks is discussed. In the other category, we study the BF applications in various fields related to wired networks, such as tracebacking, pattern matching, and so on. The last three columns of this table give information about where Bloom filters have been embedded in each specific security application. These columns indicate whether the Bloom filters used for each security field are embedded in *end-devices* (ED), such as server machine, *intermediate-devices* (ID), such as routers or sensor nodes in the sensor networks, and/or *in-packet* (IP), where Bloom filter is located inside the packet traversing the network [20]. We emphasize that this categorization is not completely exhaustive. There may be some works that could be fallen into more than one category.

3.1. Wireless networks

In this section, the BFs applications related to various kinds of wireless networks are discussed.

3.1.1. Authentication

3.1.1.1. Message authentication. In [21], Son et al. proposed a communication-efficient message authentication protocol to authenticate messages flooded in large-scale WSNs. Each sensor node is preloaded with l symmetric keys and k hash functions. The sink also maintains k hash functions and n keys. The sink constructs n *message authentication codes* (MACs) using the n keys. These resulting MACs are then inserted into the BF. Subsequently, the BF is flooded along with the message in the whole network. When the message arrives at each node, l MACs are constructed again in the same way by using l keys stored in the node. These l MACs are sought in the arrived BF. When a zero value is found, the message is assumed to be invalid; otherwise, it is sent to the neighbor nodes [21]. Moreover, they proposed to use *compressed Bloom filters* [9] for reducing false positive rate and the size of BF.

Table 1
A taxonomy of Bloom filter Application in Network Security; end-devices (ED), intermediate-device (ID), in-packet (IP).

Environment	Application		ED	IP	ID
Wireless networks	– Authentication	– Message authentication	×	√	√
		– Node authentication	×	√	√
	– Anonymity and privacy-preserving	– Anonymous routing	×	√	√
		– privacy-preserving	√	√	√
	– Firewalling	– Mesh firewall	×	√	√
		– 3G firewall	×	×	√
	– Tracebacking		√	×	×
	– Misbehavior detection		×	√	√
	– Replay protection		×	√	√
	– Node replication detection		×	√	√
Wired networks	– String matching	– Standard BF-based schemes	√	×	√
		– Counting BF-based schemes	√	×	√
		– Bloomier based schemes	√	×	√
		– Standard and counting BF-based schemes	√	×	√
	– IP tracebacking	– Logging-based IP tracebacking	×	×	√
		– Marking-based IP tracebacking	×	√	×
		– Logging- and Marking-based IP tracebacking	×	√	√
	– Spam filtering and e-mail protection	– Spam filtering	√	×	×
		– E-mail Server protection	√	×	×
	– DoS and DDoS detection	– DoS and DDoS attacks addressing	×	√	√
		– DNS attacks addressing	×	×	√
		– SYN flooding attacks addressing	×	×	√
	– Anomaly detection		×	×	√

The scheme introduced in [22] tries to protect the data of the network from the attackers in an efficient manner. In this scheme, a group of sensors, named *aggregators*, classify the packets arrived from the other sensors. This scheme utilizes BFs for keeping trade-off between communication and computation costs and also enhancing the performance of the network. To this end, BF keeps the keywords associated with the nodes in the network. The base station sends a request message in the form of $Req\ [r, BF_r(w_1, \ldots, w_n)]$ (where r is a random key and $BF_r(w_1, \ldots, w_n)$ is the BF resulting from k hash functions applied to the keyword r) either to the all nodes of the network or only to the aggregators. The sensors compare the keywords (their own BFs) with $BF_r(w_1, \ldots, w_n)$ to find a match. If there is a match, an encrypted message is generated based on the predefined policy and sent to the requester (aggregator or base station). One problem with this scheme is that the main consideration of it is to reduce the data redundancy; security is not taken into account sufficiently.

In [23], Ren et al. proposed several public key cryptography-based methods to provide a multi-user broadcast authentication service to minimize computation and communication costs. In the proposed method, called *Bloom filter-based Authentication Scheme* (BAS), all public keys assigned to the network users are inserted into a BF. Eventually, this BF is placed within all the sensor nodes of the network. When receiving a broadcast message, the sensor node checks the membership of its public key in the BF. Then, if a zero value is found, the message is discarded. The scheme is of interest but is applicable for special kind of WSNs with many user nodes. Moreover, the whole scheme cannot resist DoS attack. In addition, the long time to verify each message using PKC increases the response time of the nodes.

In [24], another source authentication scheme based on multi-level μ TESLA has been proposed in which BF is used to store MACs in order to diminish the communication cost and the total energy overhead. In this case, the sender generates d MACs for each packet in specified time intervals. The d MACs of each packet are mapped into its BF. When receiving a packet, if the number of 1's in the BF is less than or equal to $d \times k$, the receiver calculates MAC by using the corresponding key; otherwise, the packet is dropped. If the mentioned condition occurs, the resulting MAC value is sought in the BF and the valid packet is recognized. This scheme cannot eliminate the node compromise problem, which is a very difficult problem in WSNs.

In addition, the multi-user broadcast authentication scheme proposed in [25] uses BFs to address multiuser authentication problem in WSNs. This protocol is based on *Elliptic Curve Cryptography* (ECC) algorithm. In this case, the sink associates each user in the network with a public key, where $PS = \{< ID_1, PK_{ID_1} >, < ID_2, PK_{ID_2} >, \ldots, < ID_n, PK_{ID_n} >\}$ indicates the set of users and public keys. All the elements of this set are then mapped into the BF. This BF is located in each sensor node in advance. Upon receiving a broadcast message, k hash functions are applied to the $\langle ID, PK \rangle$ pair contained in the message. If all the k positions in the BF are 1, the message is considered to be valid. This scheme also uses a reputation-based randomized authentication scheme to deal with DoS attacks. The authors pointed out that the scheme is more resilient to DoS attack, and the end-to-end delay is acceptable.

In one-time sensor networks, each sensor can generate only one message during its lifetime but can always retransmit messages arrived from other sensors. To combat intrusions into such networks, a cost-efficient scheme was proposed in [26], in which BF keeps the (idx, id) pair

values of the all nodes in the network. Both the index value ($idx, 0 \ldots n-1$) and the identifier value ($id, 0 \ldots N-1$) are located in the message header, where n is the number of nodes in the sensor network and N is a large set of values, in which $n \ll N$. When receiving a message, sensor checks the membership of (idx, id) of the message in the BF. If a zero value is detected, the message is considered to be a spurious one. The authors believe that this scheme can work well against all kinds of attacks [26]. Moreover, in order to filter false messages inside the network and to eliminate bandwidth consumption, the approach proposed in [27] adds some additional information, called *En-route Authentication Bitmap* (EAB), to the messages. Instead of directly using MAC for the existing nodes in the path, the MACs are first hashed and mapped into the BF and then this new BF, i.e., EAB, is transmitted. Therefore, intermediate routers only pass the correct message by using the EAB. The authors claim that this lightweight approach has a low computational cost and a small communication overhead. However, this scheme has no resilience to the selective forwarding attack and report disruption attack.

3.1.1.2. Node authentication. In data-centric sensor networks, key BFs have been used for generating query messages and enhancing the privacy of information against various attacks [28]. In this case, in order to avoid performing membership test by the attacker, the IDs of all storage cells are encrypted using cell keys, and then hashed and inserted into the BF. In this scheme, when the query message arrives at a cell, it is sought in the key BF. If the neighbor is in the BF, the message is sent to that neighbor. This scheme can only partially address the threats against data privacy and data availability. For instance, it cannot cope with information leakage caused by node compromises or communication disturbance caused by jamming attacks. Moreover, it employs homogeneous network architecture and cannot apply to a tiered WSN. In addition, a new mechanism based on one-way functions has been proposed in [29], which employs BFs and μ TESLA to control joining and leaving of nodes in the network. In this case, the BF has been used to avoid underflow and also to protect IDs from being used further by the attackers, when leaving a member of WMSN. At the beginning, all nodes in the network are hashed into the BF. When sink detected that a node wants to leave the network, the BF is updated and sent to the all nodes in the network. Subsequently, they update their own BFs based on the new BF. However, the inherent features of such μ TESLA-like schemes, such as the need for (loose) time synchronization and the delayed authentication, have made them vulnerable to a variety of attacks.

3.1.2. Anonymity and privacy-preserving

3.1.2.1. Anonymous routing. A secure anonymous routing protocol for clustered ad hoc networks was proposed by Chen et al. [30]. Because of using BFs, this protocol does not require any public key operation. In this scheme, BF has been used to both anonymous data transmission and anonymous route discovery. The identities of the nodes in the route from source to destination are mapped into the BF. Therefore, to hide the ID, the node only needs to hash its ID by k hash functions and set the corresponding bits in the BF. In the data transmission phase, the BF containing routing information is sent along with the message. The authors believe that this protocol can provide different levels of anonymity [30]. ODAR [31] uses Bloom filters to complete anonymity of nodes and source-routing paths. However, the ODAR requires a online key distribution server G in the ad hoc network and the communication is not blind to the G. Furthermore, it only considers the source and destination pattern and not the security of the Bloom filter in the intermediate nodes.

In addition, a storage- and communication-efficient approach, called *anonymous multi-cast routing* (AMUR) for ad hoc networks was proposed in [32]. This approach benefits from the use of BF and Diffie–Hellman key exchange protocol to provide anonymous routing. In that paper, BF has been used to provide anonymity in multicast routing. To do so, BF maintains the links from transmitter to receiver. When a packet arrives at a node, the BF is sought to check the membership of the links. Moreover, another extension of BFs, called *ToPoBF*, has been introduced based on *attenuated BF* to store routing information, i.e., information about the nodes in the next hop and their distance. In general, AMUR can provide strong anonymity to data forwarding and routing control mechanisms and thwarts address spoofing attacks. However, AMUR cannot prevent nodes on the source path from injecting invalid packets and staging denial of service attacks in the ad hoc network.

3.1.2.2. Privacy-preserving mechanisms. In [33], Zhu and Mutka have proposed a message notification protocol to reduce power consumption and wireless wide area network (WWAN) access costs for *instant messaging* (IM) services that convey presence information of mobile users. In *Cooperating ad Hoc network to sUpport Messaging* (CHUM) only one of the terminals in the ad hoc network at a time, acting as proxy, needs to have access to the IM server in the fixed network. A proxy should not be able to see or change the content of the messages sent to the other peers. Usernames should also be esoteric when needed. In this situation, *compressed Bloom filter* [9] is used in CHUM to store and represent the message notification exchanged between the IM server and the peer group to provide security and privacy and also to reduce the overhead of the protocol [33]. However, if the proxy is compromised, it is difficult to detect whether the Bloom filter has been changed. In order to cope with privacy problems caused by the use of *Radio Frequency Identification* (RFID) in computing environments, Nohara et al. in [34] proposed a high-speed identification scheme in which the pre-calculated outputs of the tags are saved in the BF. The problems arise when the attacker tries to use the ID of the tags to keep track of the user. This scheme consists of three phases: pre-computing, identification and updating. For each tag_i, there is a BF_i that stores the set of the outputs of the tag. In the second phase, in order to search a specific ID, all the BFs are checked. If a match is detected, the ID is retrieved from the corresponding BF. The authors pointed out that their scheme can update the pre-calculation results efficiently and can always keep the constant margin for synchronization as compared to the other schemes such as Avoine. In [35], Yang et al.

have designed an algorithm which uses sensitive data hiding techniques inside the traversing packets to protect sensitive data in WSNs. The sensitive information is saved in BF and then this BF is located inside the main data of the packet. Therefore, the attackers will not be able to see it. Later, the destination node retrieves the sensitive information according to the predefined extraction rules. This scheme is able to hide sensitive information effectively and avoid adversaries? attention, but it cannot completely resist various attacks.

3.1.3. Firewalling

This section provides the application of Bloom filters to address the problem of firewalling. Table 2 lists the words related to the abbreviations used in this section.

3.1.3.1. Mesh firewall. In mesh networks, firewall schemes are essential to classify and filter traffic. Maccari et al. [36] proposed a scheme that uses BFs to create a distributed firewall. In the scheme, each node adopts a Bloom filter to represent all packets accepted by the node, and then distributes the Bloom filter to all nodes in the network. When a node wants to forward a packet, it queries the packet from all Bloom filters it has received from other nodes. If it is found, the packet is forwarded; otherwise, it is discarded. In this scheme, a firewall rule is presented by the set $R = \{sourceIP, destinationIP, sourcePort, destinationPort\}$. The authors consider their scheme as a stateless firewall, as it does not taken the state of the connection into account. When implementation, they only consider packets with class C IP addresses and port numbers less than 1024, which is a drawback of the scheme. In [37], Maccari et al. reported further results on the use of CBFs to address the problem of firewalling in a real-time testbed. The authors extend their work in [38] to support stateful firewalls and they use dlCBF [8] with handover support to save memory space. The scheme uses state automation to evaluate the accuracy of the relationship between peers. To do this, the current state of each flow is stored. This new BF maintains the set of tuples related to the next valid states, named *State Expectation* (SE). The authors named the scheme stateful, as it takes the state of the TCP connection into account when classifying packets. when a packet arrives at Access Point (AP), the new state is sought in the dlCBF. If it does not exist, the packet is not a member of the safe flows. To remove SE from the dlCBF while avoiding false negative, they have used beyond Bloom filters. In general, since the amount of accepted packets is huge in the mesh networks, the Bloom filter is very big, so the efficiency in the scheme is still a challenging issue. These schemes are not scalable as the size of Bloom filters is tightly dependent on the size of accepted packets. In addition, the distribution of the large BFs among all the nodes in the network results in high communication overhead.

Table 2
Full words related to the abbreviations mentioned in this section.

Specification	Value
APN	Access point name
LSF	Longest sub-string first
TSBF	Two-stage structure BF
IMSI	International mobile subscriber identity
GTP	GPRS tunneling protocol
MCC	Mobile country code
MNC	Mobile network code
MSIN	Mobile subscriber identification number
TEID	Tunnel endpoint identifier

3.1.3.2. 3G firewall. Bloom filters also have been used in 3G firewall. APN filtering and IMSI filtering are important functions in combating "create PDP request" flood attacks in 3G security devises. To perform APN filtering efficiently, a TSBF architecture was proposed in [39], which utilizes CBF [6] along with the standard BF [1] to filter APN strings. The authors reported that the performance of the TSBF is better than LSF [39]. The hardware techniques, such as LSF may be not suitable to address these problems because the maximum length of an APN string can reach 100 bytes. Moreover, the scheme proposed by Liu et al. [40] inspects GTP packets to find IMSI that matches IMSI rules stored in the Bloom filter. IMSI is a unique number dedicated to the cell phone users in 3G networks. This parallel scheme is able to match prefix and whole string by using IXP2850 network processor and BF. In addition, the scheme proposed in [41] inspects GTP channels to cope with the huge number of flows each including a large number of packets. To do so, CBFs have been used in parallel to maintain flow information in the processors. A new TEID is calculated by the hash functions, the corresponding bits of the BF are set to 1 and the counter should accumulate. All the TEIDs in the packets should be checked. A decision word is then initialized by zero, and the TEID is hashed using k hash functions. Eventually, if the TEID string is found in the BF, the packet is legal and be forwarded; otherwise, it is dropped. They have pointed out that the scheme can inspect of 1Gbps flow in the GTP channel. Although the Bloom filter can compact the data structure, the efficiency in the scheme is still a challenging issue once the amount of packets may be huge in such an environment.

3.1.4. Tracebacking

3.1.4.1. Tracebacking in WSN. In the architecture proposed in [42], cooperative sensors utilize multi-dimensional BFs, named *space–time Bloom filters*, to maintain the attributes of the packets in order to traceback the attacker packets. In this case, in addition to the packet information, the ID of the forwarding node is also added to the input string of the hash functions. When passing a packet through a sensor, this packet is mapped into the BF of the sensor. Later, the BF will be used to reconstruct the attack graph. However, this scheme has been designed for a small sensor network, and it has no feature to recompute the attack path.

3.1.4.2. Tracebacking in MANETs. In [43], Kim and Kim proposed a logging-based IP traceback technique which utilizes another extension of BFs, called *time-tagged Bloom filter*, to maintain the information of the packet passing through the router. This scheme uses the 28 bytes of the packet IP header and 8 bytes of the IP data as input to

the hash function. To control collision and to avoid including safe routers in the tracebacking process in a long-term, each entry of the BF is equipped with a time-tag with an initial value. Each node that tries to detect attack collects information and sends it to the cluster head in the form of an IP traceback message. After authentication, the message is sent to the neighbors of the node to be checked in the BF. If the answer is positive, it is sent to the other neighbors in the next hop to reconstruct the attack graph [43]. Hotspot technique proposed in [44] adds TTL-tags in the Bloom filter and uses this information together with a neighboring list to find the nodes in the attack path. Since the request for path recovery is broadcast, it causes heavy network traffic. It also does not show clearly how to refresh the Bloom filter. In addition, in the scheme proposed in [45], the authors have proposed two advancements in the previous traceback schemes such as [42,43]. The scheme uses multiple IDBFs (ID based Bloom filters) to reduce false positive rate. Also, they added the support of directed queries, which reduces the number of messages generated by the traceback process. One problem with this scheme is that it uses more memory than the traditional traceback schemes specially when the number of nodes increases in the network.

3.1.4.3. Tracebacking in WiBro. An IP traceback methodology using Markov chain and BFs for E802.16 protocol was introduced in [46], in which the duty of the BF is to store the information about the routers. This methodology verifies the normality of the hashed information and then performs IP traceback.

3.1.5. Misbehavior detection

3.1.5.1. Misbehaving node. Kozma and Lazos proposed a technique to recognize misbehaving nodes, which does not rely on continuous overhearing or intensive acknowledgment techniques [47]. In this technique, the source node *S* audits the nodes in the network in a specified time period to identify the nodes refusing to forward packets to a destination. The source node *S* sends an audit request to the suspected node and asks it to keep track of the packets sent by the suspected node during that time period. Since a huge memory is needed to store this information, the suspected node utilizes a Bloom filter to represent the set of packets forwarded. The suspected node then sends its BF to the source node. The source node then evaluates the behavior of the intermediate node. The drawback of the scheme is that it only focuses on the problem of identifying one misbehavior node and it has not been evaluated for multiple misbehaving nodes. In addition, There is no proposal to protect the Bloom filter from attacks such as burst-force one.

3.1.5.2. Misbehaving vehicle. Since the idea of using pseudonym to ensure privacy in VANETs was introduced, some malicious vehicles abused this attribute by continually changing their identities. In order to isolate a malicious vehicle in VANETs, Liu et al. [48] proposed a mechanism in which each vehicle maintains its own reputation in its Tamper-Proof Device (TPD) based on the reputation segments of its neighbors. In this case, the BF is used to record both dishonest and trusted vehicles and to reduce the overhead of message broadcasting. The authors believe that this scheme ensures both the privacy and security of data. However, this scheme fails to consider the common attacks to reputation aggregation, e.g., blocking negative reputation segments.

3.1.6. Replay attack detection

One of the most common methods for checking the freshness and thus protecting the message from replay attacks is to use sequence numbers. In [49], a protocol, called *Low-Overhead Freshness Transmission* (LOFT), has been introduced in which only the least significant bits of the sequence numbers are transmitted along with the message. In the case of abnormal increase of the arrived messages and in order to diminish the overhead of freshness check caused by DOS attacks, LOFT uses BFs to keep the last *w* messages sent by the sender. In this situation, before checking the freshness of the message, the receiver checks the recent *w* messages mapped into the BF. If the answer is not positive, the message is discarded; otherwise, the freshness of the message must be checked exactly. The authors pointed out that LOFT is more tolerant to message loss and replay attacks than the previous schemes such as SNEP. However, LOFT is not applicable for the sensor nodes that are multiple hops away. It just focuses on communication between adjacent sensor nodes. In addition, the SNEP scheme works better than LOFT in term of freshness transmission overhead. In [50], Jinwala et al. argued that any replay detection scheme in WSNs must be implemented at the link layer. They discussed and implemented three approaches, counter-based, hash-based and BF-based approaches. In the third solution, instead of the hash and counter values, BF is used to reduce the memory overhead. That is, the received packets are completely hashed and inserted into the BF. The freshness of the incoming packet is verified using this Bloom filter. They pointed out that the BF-based solution works well for different types of networks, regardless of the number of the nodes in the network. However, increasing the number of packets in the network results in the increase in false positive rate. This matter has not been clearly discussed in that paper.

3.1.7. Node replication detection

In a node replication attack, several nodes decide to use the same ID in WSN. The hierarchical algorithm proposed in [51] uses BFs to detect replicas. In this hierarchical structure, the cluster-head nodes selected by the other nodes or by *Local Negotiated Clustering Algorithm* have responsibility for detecting replicated nodes. To this end, the IDs associated with all the nodes in the cluster are mapped into the *dynamic Bloom filters* by the cluster head. Subsequently, this dynamic Bloom filter is encrypted and sent to the destination node, along with the other encryption information of the node. The receiver then searches dynamic Bloom filter to find the IDs of the nodes included in the cluster. If a match is found, it sends the matched ID to the sender cluster-head node for performing exact checking. If the answer is positive, the ID is considered as a replica [51]. The scheme proposed by Tong et al. [52] intends to broadcast intruder information to the all sensor

nodes in the network in order to address intruder replication problem properly. The scheme uses *cooperative BFs* for local management of intruder information and for saving space in sensors. To this end, each node maintains a BF containing the IDs of all detected intruders. A special server, called *dedicated membership server* (DMS), periodically sends the information of the recently detected intruders to the all nodes in the network. These nodes then add these new intruders to their own lists. Eventually, the compromised nodes are detected according to the information received from the neighbor nodes [52]. In [53], Zhang et al. presented two new techniques, which are called *cell forwarding* and *cross forwarding* to improve the node replica detection in WSNs. The proposed schemes use BFs to store the information stored at the sensors to reduce the memory usage of intermediate nodes in LSM. These schemes use two BFs, one for storing ID of the nodes (ID filter) and the other one for keeping the locations (location filter). Subsequently, these two BFs will be utilized by the nodes to detect conflicting claims in the subsequent operations. These schemes are based on distributing the location claims to relay nodes in the network. Since the location claim is distributed to many nodes in the network, it increases a chance to detect the node replication. However, these schemes have a lot of communication overhead, despite of using BFs, because they try to forward the location claims to intermediate nodes which act as a witness node. In addition, these schemes cannot detect the replication attacks in a mobile sensor environment. They rely on the relatively expensive public key cryptography.

3.2. Wired networks

In this section, we review the various uses of BFs in the design of different security mechanisms proposed for wired networks.

3.2.1. String matching

The core operation of the deep packet inspection is to search for predefined signatures in the packet payload. This is also known as *string matching*. In this section, we demonstrate where BFs and their variants have been utilized to improve the efficiency of string matching algorithms. We categorize these approaches according to the type of BF used.

3.2.1.1. Standard Bloom filter-based schemes. In [54], a set of hardware BFs have been used in parallel to verify which input flow matches against a set of predefined signatures. In this architecture, each BF maintains the signatures of a particular length. Therefore, each BF is utilized to find the strings of a specific length in the input stream. This architecture is depicted in Fig. 4. In each run, a window of the data stream is inspected by the system. If each of these BFs detects a match, the string is delivered to the analyzer to perform exact matching; otherwise, the next byte of the stream is processed. If there are multiple matches for different lengths, the longest one is selected. In contrast to the previous methods, such a Bloom filter-based system is able to handle large databases with reasonable resources and supports fast updates to the database. However, the

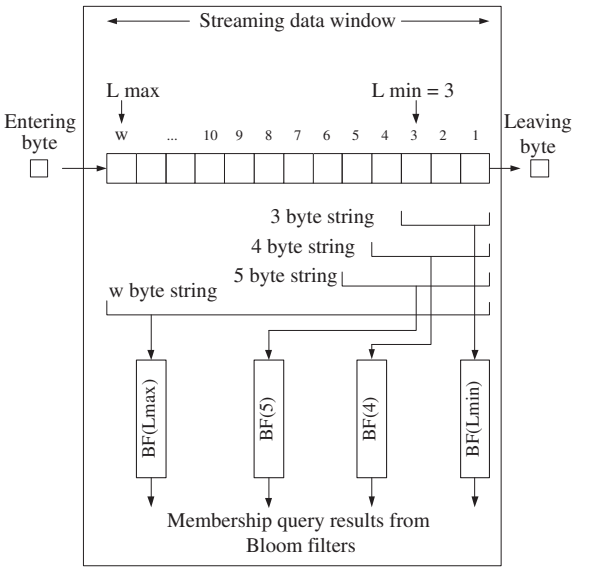

Fig. 4. A window of streaming data containing strings of length from $L_{min} = 3$ to $L_{max} = w$ [54].

analyzer might be much slower than BFs, if false positive rate is too high, additional accesses to the analyzer will reduce overall system's throughput. Lee and Choi in [55] have improved the hardware Bloom filter proposed in [54] so that it can discover the fault (which may generate some false results) caused by mistakenly being reset a specific bit when computing hash values. The implementation results of BFs on the *Field Programmable Port Extender* (FPX) platform [56] for string matching have been shown in [57].

In [54], the number of the Bloom filters increases linear with the number of various pattern lengths. In [58], in order to split long patterns into small substrings for reducing the number of Bloom filters, *stateful BF engine* has been proposed which utilizes a special prefix register heap in addition to parallel BFs and lookup table. For detecting long patterns, these parallel BFs maintain intermediate statuses, i.e., the index of the current matched substring. This is because after detecting a match in any of the engines, there is no need to check all the patterns in the set when doing exact matching in the next stage. Because long patterns are split into small substrings, the intermediate statuses need to be saved. In the second stage, both the matched substring and the active prefix are used to perform deterministic string matching [58].

The work in [54] does not focus on the software implementation and CPU computation cost aspects of hash functions used in scanning application. The Hash-AV system proposed in [59] tries to embed BFs and hash functions on the CPU second-level cache in order to use the capabilities of CPU for scanning viruses. Hash-AV utilizes two groups of hash functions: bad-but-cheap hash functions to do the approximate scan in the first stage and good-but-expensive hash functions to do the exact scan in the second stage. The second group is used only if there is a need to do exact matching. Using the bad-but-cheap hash functions, the CPU computational cost is reduced when

scanning BF because there are mostly no match in the first stage. In contrast with [54], k hash functions are not calculated simultaneously in Hash-AV. If the bit corresponding to the output value of the current hash function is 1, the next function will be computed. However, the applicability of the Hash-AV as a software solution for other string matching based application such as anti-spam applications has not been discussed in [59].

The scheme in [54] is not able to efficiently address multi-packet signature detection problem, because a BF is not capable of recognizing partial signatures. This motivates Artan and Chao to propose an architecture composed of a flow processor and a payload processor [60]. The former maintains per flow state information for multi-packet signature detection, whereas the latter uses a combination of parallel BFs. More precisely, the payload processor adopts, for each length, a BF that represents all the strings of that length, as well as a BF that represents all the string pieces of that length. Fig. 5 illustrates the proposed architecture. When a packet arrives, a complete check is performed on all the filters (an expensive process). If a match is detected, the flow database is updated, and the state becomes malicious (if a whole signature is found) or suspicious (if a simple piece is found). Whenever the flow state is malicious, the flow is passed to an analyzer for a further deterministic check. This scheme assumes that packets are not ambiguous, in order, and not overlapped, thus neglecting many real issues. Moreover, the use of filters for prefixes of one or two bytes appears too expensive for memory requirements, processing power, and alert rate, thus making such a system inefficient.

As some of the previous works, Dharmapurikar and Lockwood [61] use on-chip memory in FPGA/VLSI to fulfill high-speed pattern matching. In addition to the off-chip hash table which saves strings, on-chip BFs are utilized to store the strings on the FPGA. There is a distinct on-chip BF for each hash table. If the matching occurs in each BF, the off-chip hash table is scanned to do exact matching; otherwise, there is no need for the hash tables. As a result, the number of references to the off-chip memory is highly reduced, and the speed and performance of the system are also improved. Fig. 6 depicts the block diagram of the system proposed in [61].

In [62], Artan et al. wanted to improve query throughput and solve the memory fragmentation problem caused by using several BFs by NIDS/NIPSs to deal with small sets. To do this, they proposed *aggregated BF* in which queries are hashed in sequence and then aggregated to enhance the average throughput. The proposed architecture is depicted in Fig. 7.

The m-bit BF is divided into k portions where each portion corresponds to a hash function, leading to form a function-bitmap pair, called *Processing Element* (PE). Each PE_i is responsible for a query queue. For each query Q_i, the counter (C_i) counts the number of matches. Each PE_i processes the first query in the queue. If the matching occurs for Q_A, the counter is increased and $PE_{(i+1)}$ will process Q_A; otherwise, Q_A is discarded. All the matched queries and the corresponding counters are delivered to the next PEs. To address various signature lengths, several sets are aggregated in one m-bit BF. This aggregation can reduce the overall number of queries. The authors claim that the aggregated BF represents sevenfold improvement in the

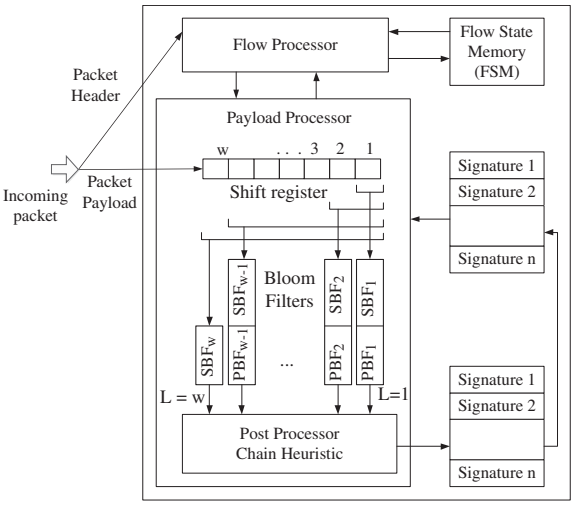

Fig. 5. Multi-packet signature detection system ($W = L_{max}$) [60].

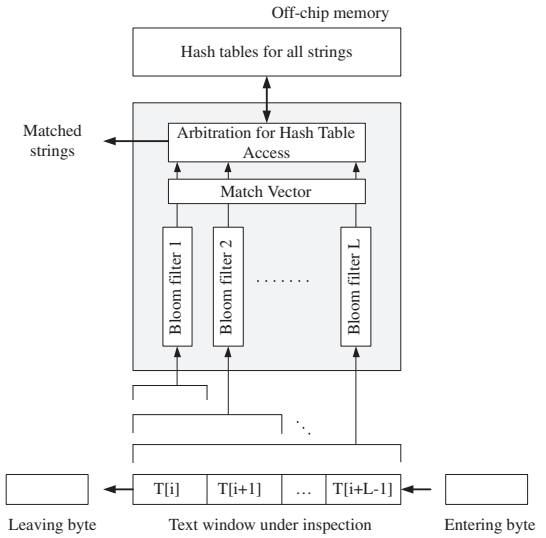

Fig. 6. A string matching machine consisting of multiple Bloom filters each of which detects strings of unique length [61].

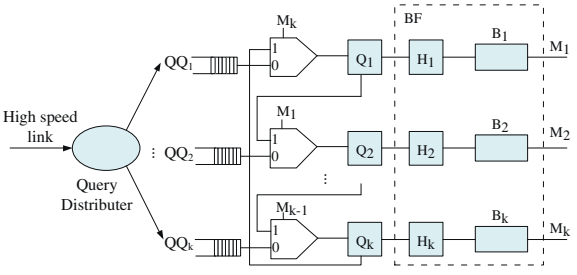

Fig. 7. Hardware architecture to support aggregated queries [62].

average query throughput and four times less memory usage compared to the previous hardware BFs for this application [62].

In [63], Nourani and Katta proposed another architecture based on BFs and *Parallel Hashing* (PH), in which BF acts as an accelerator and does preliminary matching. In this case, an *l*-byte substring of the input stream is hashed and sought in the BF. If a match is detected, the dispatcher sends this substring to the PH engine for exact match. In comparison with the work in [54] which uses 35 hash functions, this architecture uses only one or two hash functions.

Nourani and Katta in [63] believe that their architecture can perform matching of 16,000 strings and achieves throughput in excess of 100 Gbps. Note that it is possible that the packet payload is distributed between several packets along the path. These partitioned packets also may be distanced from each other by the packet of other streams. Therefore, there is a need to embed a string matching algorithm in router, which considers the statuses of the pattern matcher when performing pattern matching. To this end, Kumar [64] has used the linearly recursive hash functions in BFs in such a way that the new hash value is calculated based on the previous value. When receiving a new packet, its stream is detected and then the hash values of the stream are loaded to be utilized in the new computations. Fig. 8 depicts the architecture. The author also purposed a string matching algorithm using *Programmable Ethernet Interface Card* (PEIC) to enhance the throughput of NIDSs at high packet rates via discarding unwanted packets [65]. This string matching is performed in the BF-equipped FPGA-based PEIC [65].

The byte-filtered string matching algorithm proposed in [66] tries to address unnecessary state transitions problem of bit-split string matching algorithm. In this case, each byte of the input stream is processed by using BFs just before performing bit-split matching. If the matching occurs, this stream of bytes is divided into a set of k-bit substrings to run bit-split string matching algorithm by parallel tiny DFAs. If a match is detected, every DFAs can make a transition to a next state, and then the output vectors are generated. Eventually, the matched strings can be detected by bitwise AND unit [66].

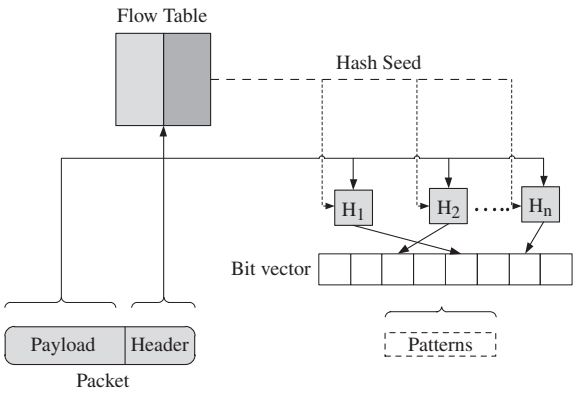

Fig. 8. Architecture overview [64].

In [67], Lin et al. proposed an architecture to perform string matching in sub-linear time based on algorithmic heuristics. This architecture uses the capabilities of parallel BFs to speed up matching operation. In this case, the patterns are divided into some groups according to their lengths and positions, and then saved in the BFs. Then, a value in the search window, called *shift value*, is specified and sought in a set of BFs. If a match is detected in any BF, it is a member and then the shift value is adjusted in accordance with the heuristics [67].

3.2.1.2. Counting Bloom filter-based schemes. The pipeline architecture proposed by Kefu et al. [68] puts to use the benefits of CBFs to perform deep packet inspection. This system consists of two sub-systems: the fast pipeline sub-system which carries out approximate checking to detect suspected substrings and low pipeline sub-system which exactly checks the output substrings of the first sub-system. In contrast with the previous schemes such as [54,57,60], this architecture is a loosely coupled framework in which the approximate matching is performed asynchronously from the exact matching operation. As a result, the speed of the architecture is more than that of previous systems [68].

Another FPGA-based architecture introduced for intrusion detection problem puts into service the capabilities of FPGAs to implement CBFs to support insertion and deletion operations of viruses and worms [69]. This architecture has been called *system on-a-chip* because of the presence of power-pc 405 processor and thus there is no need for any extra computer to establish network communications. 2-, 4-, 8-byte data inputs are employed, in turn, by filters and the results of their querying in the CBFs are analyzed by the power-pc processor. The real threat is detected with the help of an extra hash table placed in DDR-RAM of the FPGA and eventually the packet is dropped [69].

Based on *Parallel Longest Prefix Matching* (PLPM) [54] and *Longest Prefix Matching* (LPM) [70], a new architecture, called *Memory Efficient Parallel Bloom filter* (MEPBF), was proposed in [71] that consumes less memory than PLPM. This is because only one 2-bit counter is utilized in their proposed design. However, the speed of string matching process in the architecture is less than that in [54].

In [72], CBFs have been used to design an anti-evasion string matching approach and also avoid reassembling the packet at high-speed rates. In fact, CBFs checks various substrings of the real string. In this architecture, the input streams are separated based on the transition protocol and fed into the substring detectors made of subCBFs. Each subCBF maintains common attacks related to each protocol. This architecture has been designed for supporting substrings of 3-byte lengths. After finding a preliminary match in the subCBF, other low-speed units, called PMEs, perform full matching. The authors claim that this approach can recognize up to 99% of attacks. However, this mechanism needs extra operation for two- or three-byte long packets. Moreover, there is no clear report about the speed of the system and also memory consumption.

In [73], instead of CBF, Lin et al. have used dlCBF [8] for performing dynamic pattern matching. In the authors

point of view, CBF has some shortcomings, such as high false positive rate, limited rule capacity and low memory utilization, which can be improved by dlCBF. It has been reported that dlCBF saves 56 times memory allocation than CBF. However, there is no analysis on the time complexity of the given algorithm.

3.2.1.3. Bloomier filter-based schemes. In [74], Ho and Lemieux have proposed a FPGA-based software architecture to fulfill pattern matching on ClamAV [75]. The ClamAV is the most popular open source anti-virus database which uses Bloomier filter [10] as the core of its architecture. The architecture consists of several predefined Bloomier filter units each of which hashes strings of a certain length into the corresponding hash table. All of the patterns are mapped into *Bloomier Filter Units* (BFUs). In each cycle, one byte of the input stream is scanned by the BFUs. If a match is detected, the information of the matched patterns is sent to another unit named *metadata unit*. This unit then extracts full information of the suspected pattern from off-chip memory and sends it to another unit to do exact matching [74]. In addition, Tuan et al. [76] offered another architecture to accelerate the performance of pattern matching in ClamAV database. The architecture reduces off-chip memory access time. This architecture combines standard BF and Bloomier filter to minimize memory access times in the comparison phase. The authors believe that the architecture can provide a significant improvement in terms of memory requirement [76]. None of these two approaches are not able to discover unknown viruses in the current design.

3.2.1.4. Standard and counting Bloom filter-based schemes. Song and Lockwood proposed a new architecture which uses a novel Extended BF (EBF) and link lists in order to optimize searching process [77]. Every bucket in the BF consists of 3 fields. The first field is one bit and has the same definition as in the standard Bloom filter. The second field counts the number of signatures hashed into the corresponding cell and the third field is a pointer which helps to store the actual items in the signature set. However for each item, it is stored for k times. In this architecture, in contrast with the work presented in [54], if there is a match in the main BF, only the shortest list is searched. The authors also proposed a scheme to deal with long signatures. They claim that this architecture can work well in terms of memory storage and throughput. However, the capability of this algorithm for scanning traffic in a high speed network has not been proven.

In addition, Shenghua et al. in [78], proposed a cascade hash design of BFs to be used in signature detection applications. This design consumes a small space while greatly reducing the false positive rate in query phase. The architecture utilizes *primary* and *secondary* BFs. The m-bit array of primary BF is split into w-size blocks and also k hash functions are divided into two groups: odd and even. Both of these addresses are placed in the same memory I/O block. The concatenation of the acquired addresses in primary BF is generated for the signature S in the form of a string. This string is hashed as a mirror image of S and then it is inserted into the secondary BF. This design has a lower false positive rate and memory consumption than the structure in [77].

To tackle worm attacks in high-speed LAN networks, Chen et al. [79] have used *parallel BFs* for building an IXP processor-based software system which works similar to the work presented in [54]. The main idea is to find and locate worms by detecting the signatures of worms in every packet enroute.

Moreover, Chen et al. [80] have introduced a new BF-based architecture to speed up membership queries by reducing memory accesses. This architecture intends to create a relationship between hashed memory addresses. To this end, it uses burst-type data I/O capabilities in DRAM design. In each insertion, 2 bits in the BF are set to 1 s. To check the membership of the element, 2 bits of each block are checked. Because every 2 bits placed in a same block shares one initiation address, these two bits are loaded together in each I/O operation. Consequently, the total number of memory I/Os is reduced by half. Therefore, the average query delay can be reduced significantly. However, its false positive rate is higher than standard Bloom filter.

There are some work trying to reduce power consumption in the current BF-based architectures [81,82]. They divides k hash functions into two groups. In [81], where the well-known pipelining technique is used, the *primary* functions are always calculated but the *secondary* functions are employed only when a match is detected in the first step. In [82], in the first phase, r hash functions out of k–r are calculated. When a zero value is detected, the other k–r hash functions are ignored [82]. As a result, Both reduce the overall power consumption of the BF.

3.2.2. IP tracebacking

In this section, we present the use of BFs in traceback schemes proposed to reconstruct attack graph.

3.2.2.1. Logging-based IP tracebacking. The *Source Path Isolation Engine* (SPIE) presented in [83–85] is a logging-based single packet IP traceback system designed for IP version 4 and 6. In SPIE, routers store packet digests, instead of packets themselves, in a Bloom filter. This BF is paged out before it becomes saturated, preventing unacceptable false-positive rates. For each arriving packet, the SPIE uses the first 24 invariant bytes of the packet (20-byte IP header with 4 bytes masked out plus the first 8 bytes of the packet payload) as input to the hash functions. In the case of IPv6, the extension header fields and initial 20 bytes of the payload are also appended to the hash input [84]. When receiving a request at SPIE for tracking the attacker's packet, the related fields are hashed and sought in the BF belonging to that particular time period. This operation is continued until the attack graph is generated. As shown in [83], the storage overhead is reduced significantly (down to 0.5% of the total link capacity per unit time). However, at routers with high speed links, the storage requirement of 0.5% of the total link capacity per unit time may be still prohibitive. Moreover, SPIE needs to examine more BFs to cover a period long enough to offset the timing uncertainties. This increases the complexity of implementation and reduces the reliability of results. Some work

have been proposed to further reduce the storage overhead of SPIE architecture [86–88].

The scheme in [86] takes the digests of the packet aggregation units, such as flow and source–destination set, providing useful capabilities with much smaller memory requirements than that of individual packet digesting [86]. But, due to increased diversity of applications, this mechanism cannot greatly reduce the memory requirement at the routers, especially when the number of flows is very high. Li et al. proposed another system that reduces memory requirements through sampling and logging only a small fraction of packets in the BF and 1 bit packet marking is used in their sampling scheme. Their simulation results showed that the system can provide a high accuracy and deal with a wide range of attacks [87]. However, because of the low sampling rate, the scheme is no longer capable to trace one attacker with only one packet. Another scheme, called *Payload Attribution System* (PAS), was designed based on a *hierarchical BF* [88] to address the problem of log-based IP traceback systems's large storage space requirement. Compared with SPIE which is a packet digesting scheme, PAS only uses the payload excerpt of a packet. It is useful when the packet header is unavailable. However, the excerpt must be long enough to identify different packets, and thus the attackers may avoid detect by attacking through a lot of packets with short payload.

In the previous schemes such as SPIE, when a router receives a query, it checks its BF for that time interval, if the result is positive, the router queries all its upstream routers, leading to a lot of unnecessary queries sent to innocent routers. To reduce the number of necessary queries and false positive rate in SPIE, a BF-based topology-aware single packet IP traceback system, called *TOPO*, was introduced in [89]. The idea of TOPO is that some routers use the packet information together with the predecessor identifier as input to the hash functions. These routers are equipped with Bloom filters. TOPO generates the packet graph based on the responses from the queried TOPO-equipped routers. It has been proposed to apply *compressed BF* [9] and *hierarchical BF* [88] to enhance the performance of the system.

The scheme proposed in [90] is based on SPIE. As the authors pointed out, because of the use of the packet's TTL field as input to the hash functions, the precision of traceback is improved. Furthermore, the number of queries employed by the traceback operation is decreased. In comparison with the above schemes, the proposed scheme has taken into account the privacy of packet information [90]. In [91], Tang et al. have introduced a traceback-based mechanism to diminish the effect of false positive when tracing a packet. This scheme utilizes statistical information of packets along with the *Traceback BF* (TBF) representing the *IP's TTL* fields of the packets to detect and block the area in which the number of attacks is high. In contrast with standard Bloom filter, each hash function in the TBF maps elements to a discrete BF, leading to the decrease in the false positive rate [91].

3.2.2.2. Marking-based IP tracebacking. A Bloom filter in an IP traceback scheme is carried by the packet traveling in the network. Takurou et al. [92] proposed a packet marking method which aims to trace a single packet without any need to a large-capacity high-speed memories. Each packet traveling in the network carries a BF keeping the information about the routers that process the packet. Each router in this scheme deterministically generates a Bloom filter of it's IP address and accumulates it with the main Bloom filter inside the IP header of the packet. The source path is then reconstructed using this information. However, the false positive rate can be rapidly increased due to the accumulation of BFs, which makes it unable to scale to large-scale DDoS attack. In [93], a fast traceback scheme was proposed based on *space-code Bloom filter* [11] to scale to large-scale DDoS attacks at high-speed links. In this scheme, in contrast with the previous work, the router information is probabilistically inserted into the Bloom filter integrated into the header of passing packet. If the BF becomes full, it is replaced by a new one so that another packet with the same source address and destination address is generated to carry the new Bloom filter. It was shown that the false positive rate, the overhead of the network and the number of required packets for reconstruction are reduced [93]. A problem with such schemes is related to the security issues such as all-one attack in which the attacker sets all bits in the BF to one. In [94], Laufer et al. proposed an IP traceback scheme, which addresses the issue of faked identification field by attacker and is able to traceback a single-packet DDoS attack. But, since it requires relatively large-size bit field (192 bits) to be included to every IP packet, and suffers from limited scalability, its practical deployment in real Internet is problematic. This scheme uses *Generalized Bloom Filter* (GBF) [15] to store the IP address of traversed routers and to avoid digest spoofing and also rendering all-one attack. In GBF, the bits of the BF array can be reset when inserting new elements [94].

3.2.2.3. Logging- and marking-based IP tracebacking. Instead of logging every packet that is traveling through the network, the scheme proposed in [95] logs only packets, which are destined to a small fraction of nodes. The scheme is based on deterministic packet marking and logging, which tends to reduce the storage requirement by logging only the packets traversing through these nodes. This scheme uses BF to maintain the *Protected Node Set* (PNS) in the form of a *Log Table* (LT). Each entry of this table contains neighbor list BF and packet BF. When a packet arrives at a *Traeback-Enabled Router* (TER), TER checks the membership of the packet in PNS based on the destination IP address. If a match is found, the packet is forwarded without any logging; otherwise, the validity of the ID will be checked. If it is valid, the ID of the previous TER through which the packet has traversed, is logged in the neighbor list BF. Then, the flow information of the packet is stored in the packet BF [95].

3.2.3. Spam filtering and e-mail protection

BFs also have been used to protect personal e-mails and to combat unwanted emails. This section describes these applications.

3.2.3.1. Spam filtering. Signature-based Collaborative Spam Detection (SCSD) systems usually maintain a huge database

containing email signatures, demanding lots of resource in signature lookup and storage. In [96], Yan and Cho have used BFs to enhance two popular SCSD systems, i.e., *Distributed Checksum Clearinghouse* (DCC) [97] and *Razor* [98]. Razor utilizes BF to maintain the database of the spam messages and DCC maintains the number of occurrence of a signature by using CBF [6] for detecting a spam signature. The authors reported that BF can significantly reduce the size of a signature database and make the signature look-up-time constant [96].

In [99], an approximate method was proposed to speed up spam filter processing. It utilizes BFs in two techniques, called *approximate pruning* and *approximate lookup*. In the former case, an m-bit BF is used to maintain the tokens resulting from parsing each message in order to reduce the delay of searching repeated tokens when performing approximate membership test. In the latter case, a two-dimensional BF is used to reduce memory requirement by supporting information retrieval. The authors reported that the scheme has shown a factor of 6x speedup with similar false negative rates and identical false positive rates compared to the original filters [99].

Takesue [100] has used local filtering in user side to detect spam messages based on the user's interest. In this way, the system utilizes two BFs merged into a single one (for reducing cache miss ratio), called *twin-BF* (TBF), which creates a blacklist of the previous spam messages and newly added spam messages by user. *Least Recently Matched* (LRM) spam messages are saved in the primary BF and *Recently Matched* (RM) spam messages in the secondary BF. In this scheme, to block a polymorphic spam attack, a partial matching is performed based on the fingerprints of k portions of each e-mail's content [100].

3.2.3.2. E-mail server protection. Most users maintain their emails in a central server to use some services, such as remote access and backup operation. In [101], a system, called *Secure Searchable Automated Remote Email Storage* (SSARES), was proposed which allows privacy-preserving search of the email server. This system utilizes a combination of *Public Encryption with Keyword Search* and Bloom filters. The keywords of the incoming email are extracted and encrypted using PEKS public key and then stored in the Bloom filter. The Bloom filter intentionally yields a high false positive rate to protect email from dictionary attacks [101].

3.2.4. DoS and DDoS attacks detection

This section presents the uses of BFs for addressing three important types of flooding attacks, including DDoS, DNS, and SYN flooding attacks.

3.2.4.1. DoS and DDoS attacks addressing. Some of the works mentioned in the previous sections, such as [86,90,92–94,96], could also be placed in this section. Various forms of DDoS attacks have led to an increased need for techniques to analyze and monitor network traffic. IDR [102] is one of the first destination addresses-based monitoring schemes, which aims to detect DDoS attack by using BFs. The IDR splits the destination address of the packet into four fields detached by dot, independently hashes them using k hash functions and then the corresponding cells in the BF are increased by 1. If the counter values exceed a predefined threshold, the packet is considered to be an attacker [102]. However, the counter value may have been increased due to inserting IP address of the other packets, leading the system to mistakenly consider the packet to be an attack traffic. To address this problem, the scheme proposed in [103] provides a relation between address fields. The scheme utilizes an IP Bloom filter module to maintain address fields separately and an extra table to manage relation among different fields. It reduces wrong detection rate. If the number of a packet exceeds a predefined threshold, this situation will be reported to the next module, called statistic module, as an abnormal traffic. In this scheme, a central controller makes the final decision.

In [104], a router-based algorithm was introduced to combat DDoS attacks. It utilizes only a small number of routers for detecting abnormal traffic. In this case, BFs create a complete list of the valid IP addresses in order to reduce memory requirements. If a host is a member of the BF, the router sends the packet to the destination; otherwise, the packet is sent to a module, called *GA-filter* for filtering bad traffic. Moreover, Rothenberg et al. [105] have proposed a secure packet forwarding mechanism which uses BFs to maintain and update identifiers of the links. When the packet arrives at a node, all the outgoing links are computed and sought in *in-packet BF*, according to the information contained in the packet, such as flow ID, keyword, and link ID. If a match is detected, the packet is forwarded through the matched link.

In [106], Du and Nakao proposed a defense architecture, called *Network Egress and Ingress filtering* (NEIF), which adopts packet symmetry as the criteria to combat DDoS attacks. Ingress filtering blocks DDoS attacks rendered by the customers and egress filtering protects the customer against attacks. Their idea is to design a memory-efficient system with low implementation complexity to be embedded in ISP edge routers. In this case, only a small number of the large flows, which are most likely responsible for the occurrence of attacks, are tracked. These major traffic flows are measured by Bloom filters through applying multiple hash functions to the ID of the flows. Consequently, suspected flows are detected based on a certain formula.

3.2.4.2. DNS attacks addressing. With increasing the IP-spoofed requests forwarded to the DNS servers, the possibility of occurrence of the DNS amplification attacks is increased. In [107], a low-cost hardware approach consisting of two phases has been proposed to deal with such attacks. In the detection phase, the attack traffic is detected. In the second phase, the scheme distinguishes the forged responses from the secure packets by using two BFs which alternately store the requests in two continuous time periods. If the input response does not match a request in the two BFs, the response is illegitimate. The authors reported that this scheme is feasible to be employed at high speed links [107].

3.2.4.3. SYN flooding attacks addressing. In [108], a *symmetric connection detection* (SCD) method has been proposed for filtering network traffic. It uses two CBFs to manage TCP SYN packets in order to detect fully established

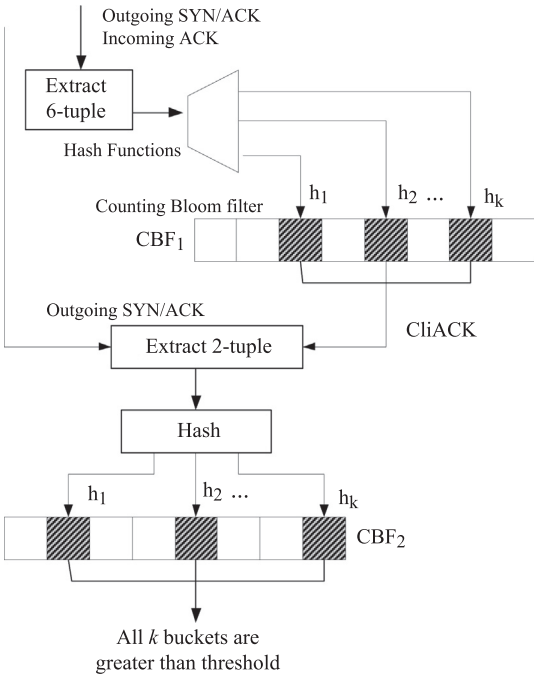

Fig. 9. The logic architecture of SACK [111].

connections. To do this, BF maintains the state of TCP connection attempts. The SCD allows only fully established flows to pass. Each one of the two CBFs is responsible for keeping SYN information in a specific direction. When a SYN packet arrives, the CBF which is responsible for the other direction is tested. If a match is detected, it means that the connection has been recently established. A new flow is inserted into the corresponding CBF. Their simulations show that the SCD can achieve accuracy of 99% [108].

The authors in [109] reported that their scheme proposed to detect SYN flooding agents can address any type of IP spoofing. This scheme utilizes SYN–SYN/ACK pairs and header information of the packets in order to detect abnormal distribution of the packets in the network. The CBF is used to classify incoming SYN–ACK packets into two groups: the first SYN/ACK packets (SYN/ACKf) and the retransmission SYN/ACK packets (SYN/ACKr). The ISN of the arriving SYN packet is hashed by and then the corresponding counters in the CBF are incremented by 1. When receiving a SYN/ACK packet, its ACK number is hashed and sought in the CBF. If all the corresponding bits are not zero, it is categorized as a SYN/ACKf packet; otherwise, it is categorized as a SYN/ACKr packet. Eventually, the attacks are detected according to the difference between the number of these two types of packets [109]. In addition, Sun et al. [110] proposed a router-based SYN flooding attack detector method which works based on the behavior of SYN–FIN or SYN–ACK pairs. Moreover, it takes the flow information of the SYN packets into account. A valid SYN packet in this scheme is a packet which closes the TCP connection. The method maintains valid SYN packets in CBF by hashing 4-tuple {source and destination port, source and destination IP} of the packet as a single item. The arriving FIN packet is sought in the CBF. If it is found, the number of valid FIN packets is increased and the item is removed from the CBF; otherwise, the packet is invalid [110]. The idea of the work is interesting but it would miss some elaborated SYN flood attacks, especially when the attack is suitably spoofed to appear benign. The authors in [111] concentrate on the accurate and fast router-based detection method for all kinds of SYN flood attacks. SACK uses Client ACK (CliACK) packets to detect SYN flooding attacks. In contrast with the previous work, SACK applies SYN/ACK–CliACK pair to detect the victim server. Fig. 9 depicts the SACK architecture. Two CBFs are used to maintain the full information of TCP connection, including the 6-tuple of the output SYN/ACK packet, i.e., source and destination's IP addresses, source and destination's ports, sequence number and ACK sequence number, and also the same 6-tuple of the input ACK packet. The authors reported that

Table 3
Bloom filter variants and their contribution to network security; false positive (FP), false negative (FN).

Bloom filter	FP	FN	Security usage	Application domain
Standard Bloom filter	Yes	No	Yes	Authentication, Firewalling, Anomaly detection, Tracebacking, Node replication detection, Anonymous routing and privacy-preserving, String matching, DoS and DDoS addressing, Email protection, Misbehavior detection
Adaptive Bloom filter	Yes	No	No	
Bloomier filter	Yes	No	Yes	String matching
Compressed Bloom filter	Yes	No	Yes	Authentication, IP tracebacking
Counting Bloom filter	Yes	No	Yes	Firewalling, String matching, Email protection, SYN flooding addressing
Decaying Bloom filter	Yes	No	No	
Deletable Bloom filter	Yes	No	No	
Distance-sensitive Bloom filters	Yes	Yes	No	
Dynamic Bloom filter	Yes	No	Yes	Node replication detection
Generalized Bloom filter	Yes	Yes	Yes	IP tracebacking
Hierarchical Bloom filter	Yes	No	Yes	IP tracebacking
Retouched Bloom filter	Yes	Yes	No	
Scalable Bloom filter	Yes	No	No	
Space Code Bloom filter	Yes	No	Yes	IP tracebacking
Spectral Bloom filter	Yes	No	No	
Split Bloom filter	Yes	No	No	
Stable Bloom filter	Yes	Yes	No	
Weighted Bloom filter	Yes	No	No	

in the worst case, the memory cost of SACK for 10 Gbps links is about 364 KB, which makes it well-suited to be embedded in modern routers [111].

3.2.5. Anomaly detection

In [112], Patcha and Park proposed a network anomaly detection system based on stochastic clustering of the network flows. Incoming audit data is clustered based on expectation–maximization (EM) algorithm. In this case, BF is used to accelerate convergence of the clustering process. In this scheme, the cluster candidates are hashed and inserted into the BF. Then, if the value of each entry of the BF is greater than or equal to a threshold value, a new cluster is created. Subsequently, these clusters are applied to detect anomaly. The authors reported that their scheme can detect anomaly, with high accuracy, even when complete audit data is not available. In addition, an extension of Bloom filters, called *Bloom filter Array* (BFA), has been used in [113] to efficiently extract two-directional (2D) matching features from traffic in order to help anomaly detection systems. The authors reported that this algorithm needs a memory of only 62.9 Mbits at the cost of losing 1% accuracy in feature extraction, compared to the 1.01 Gbits of the other algorithm (i.e., hash table) [113].

4. Summary

In the last decade, Bloom filters have received a great attention in the network security area. This is because of their key features such as low memory requirement, high processing speed, low implementation complexity and the probabilistic nature of them. In this work, we provided an updated and comprehensive survey of the application of Bloom filters in various security application in both wired and wireless networks. Table 3 summarizes the contribution of various types of Bloom filters introduced in this paper to network security. For each variant, this table indicates its application domain and whether the false positives (FP) and/or false negatives (FN) are introduced (Yes/No). We believe that Bloom filters will continue to be used in many new applications and also next variants of this structure will be introduced to deal with the incoming security problems.

References

[1] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communication of the ACM 13 (7) (1970) 422–426.
[2] M.K. James, Optimal semijoins for distributed database systems, IEEE Transactions on Software Engineering 16 (1990) 558–560.
[3] M.D. McIlroy, Development of a spelling list, IEEE Transactions on Communications 30 (1982) 91–99.
[4] L.L. Gremillion, Designing a bloom filter for differential file access, Communications of the ACM 25 (1982) 600–604.
[5] A. Broder, M. Mitzenmacher, Network applications of bloom filters: a survey, Internet Mathematics 1 (2003) 485–509.
[6] L. Fan, P. Cao, J. Almeida, A.Z. Broder, Summary cache: a scalable wide-area web cache sharing protocol, IEEE/ACM Transactions on Networking 8 (3) (2000) 281–293.
[7] S. Tarkoma, C.E. Rothenberg, E. Lagerspetz, Theory and practice of bloom filters for distributed systems, IEEE Communications Surveys and Tutorials 14 (1) (2012) 131–155.
[8] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, G. Varghese, An improved construction for counting bloom filters, in: Proceedings of the 14th Conference on Annual European Symposium, vol. 14, 2006, pp. 684–695.
[9] M. Mitzenmacher, Compressed Bloom filters, IEEE/ACM Transactions on Networking 10 (5) (2002) 604–612.
[10] B. Chazelle, J. Kilian, R. Rubinfeld, A. Tal, The Bloomier filter: an efficient data structure for static support lookup tables, in: Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, 2004, pp. 30–39.
[11] A. Kumar, J.J. Xu, J. Wang, O. Spatschek, L.E. Li, Space-code bloom filter for efficient PerFlow traffic measurement, in: Proceedings of Conference of the IEEE Computer and Communications Societies, INFOCOM, vol. 3, 2004, pp. 1762–1773.
[12] D. Guo, J. Wu, H. Chen, X. Luo, Theory and network applications of dynamic Bloom filters, in: Proceedings of 25th IEEE INFOCOM, 2006, pp. 1–12.
[13] A. Kirsch, M. Mitzenmacher, Distance-sensitive Bloom filters, in: Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments, 2006.
[14] S. Cohen, Y. Matias, Spectral Bloom filters, in: Proceedings of 22th ACM SIGMOD, 2003, pp. 241–252.
[15] R.P. Laufer, P.B. Velloso, O.C.M.B. Duarte, Generalized Bloom Filters, Tech. rep., Univ. of California, Los Angeles (UCLA), 2005.
[16] P.S. Almeida, C. Baquero, N.M. Preguica, D. Hutchison, Scalable Bloom filters, Information Processing Letters 101 (6) (2007) 255–261.
[17] M. Xiao, Y. Dai, X. Li, Split Bloom filters, Chinese Journal of Electronic 32 (2) (2004) 241–245.
[18] S.C. Rhea, J. Kubiatowicz, Probabilistic location and routing, in: Proceedings of IEEE INFOCOM, 2004, pp. 1248–1257.
[19] F. Hao, M. Kodialam, T.V. Lakshman, Incremental Bloom filters, in: 27th IEEE Conference on Computer and Communications, INFOCOM, 2008, pp. 1067–1075.
[20] C.E. Rothenberg, C.A. Macapuna, F.L. Verdi, M.F. Magalhes, A. Wiesmaier, In-packet Bloom filters: design and networking applications, Computer Networks 55 (6) (2011) 1364–1378.
[21] J.H. Son, H. Luo, S.W. Seo, Authenticated flooding in large-scale sensor networks, in: IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS), 2005, pp. 536–543.
[22] T. Li, Y. Wu, H. Zhu, An efficient scheme for encrypted data aggregation on sensor networks, in: IEEE 63th Vehicular Technology Conference, vol. 2, 2006, pp. 831–835.
[23] K. Ren, W. Lou, Y. Zhang, Multi-user broadcast authentication in wireless sensor networks, in: 4th Annual IEEE Communications Society on Sensor, Mesh, and Ad hoc communications and Networks, 2007, pp. 223–232.
[24] W.B. Jaballah, A. Meddeb, H. Youssef, An efficient source authentication scheme in wireless sensor network, in: IEEE/ACS International Conference on Computer Systems and Applications (AICCSA), 2010, pp. 1–7.
[25] X. Gan, Q. Li, A multi-user dos-containment broadcast authentication scheme for wireless sensor networks, in: IEEE International Conference on Information Technology and Computer Science (ITCS), vol. 1, 2009, pp. 472–475.
[26] C. Gamage, J. Leiwo, K. Bicakci, B. Crispo, A.S. Tanenbaum, A cost-efficient counter-intrusion scheme for one-time sensor networks, in: International Conference on Intelligent Sensors, Sensor Networks and Information Processing Conference, 2005, pp. 45–50.
[27] Y.S. Chen, C.L. Lei, Filtering false messages en-route in wireless multi-hop networks, in: IEEE Wireless Communications and Networking Conference, 2010, pp. 1–6.
[28] M. Shao, S. Zhu, W. Zhang, G. Cao, Y. Yang, pDCS: security and privacy support for data-centric sensor networks, IEEE Transactions on Mobile Computing 8 (8) (2009) 1023–1138.
[29] Y. Jia, B. Sun, Q. Zhu, A strategy of node membership verification for wireless multimedia sensor networks, in: 5th IEEE International Conference on Wireless Communications, Networking and Mobile Computing, WICOM, 2009, pp. 1–4.
[30] S. Chen, L. Xu, Z. Chen, Secure anonymous routing in trust and clustered wireless ad hoc networks, in: Second International Conference on Communications and Networking in China, 2007, pp. 994–998.
[31] D. Sy, R. Chen, L. Bao, ODAR: on-demand anonymous routing in ad hoc networks, in: IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS), 2006, pp. 267–276.
[32] L. Bao, A new approach to anonymous multicast routing in ad hoc networks, in: Second IEEE International Conference on Communications and Networking in China, 2007, pp. 1004–1008.
[33] D. Zhu, M. Mutka, Sharing presence information and message notification in an ad hoc network, in: Proceedings of the First IEEE

International Conference on Pervasive Computing and Communications (PerCom), 2003, pp. 351–358.
[34] Y. Nohara, S. Inoue, H. Yasuura, A secure high-speed identification scheme for RFID using bloom filters, in: Proceedings of the 2008 Third International Conference on Availability, Reliability and Security, 2008, pp. 717–722.
[35] J. Yang, X. Sun, B. Wang, X. Xiao, X. Wang, D. Luo, Bloom filter-based data hiding algorithm in wireless sensor networks, in: 5th International Conference on Future Information Technology (FutureTech), 2010, pp. 1–6.
[36] L. Maccari, R. Fantacci, P. Neira, R.M. Gasca, Mesh network firewalling with Bloom filters, in: IEEE International Conference on Communications (ICC), 2007, pp. 1546–1551.
[37] L. Maccari, P. Neira, R. Fantacci, R. Gasca, Efficient packet filtering in wireless ad-hoc networks, IEEE Communications Magazine 46 (2) (2008) 104–110.
[38] P. Neira, R.M. Gasca, L. Maccari, L. Lefevre, Stateful firewalling for wireless mesh networks, in: IEEE New Technologies, Mobility and Security, 2008, pp. 1–5.
[39] Z. Liu, S. Xie, Y. Lai, A fast bloom filters method in APN filtering, in: Pacific-Asia Workshop on Computational Intelligence and Industrial Application (PACIIA), 2008, pp. 145–150.
[40] Z.Y. Liu, S. Xie, Y. Yue, A parallel method in the 3G firewall, in: Proceedings of the 2009 WRI International Conference on Communications and Mobile Computing, 2009, pp. 502–506.
[41] Z.Y. Liu, W. Li, Y. Lai, Application of bloom filter for GTP stateful inspection in network processor, in: Proceedings of the 2009 Fifth International Conference on Information Assurance and Security, 2009, pp. 589–592.
[42] D. Sy, L. Bao, CAPTRA: coordinated packet traceback, in: Proceedings of the fifth international conference on Information Processing in Sensor Networks, 2006, pp. 124–135.
[43] I.Y. Kim, K.C. Kim, A resource-efficient IP traceback technique for mobile ad-hoc networks based on time-tagged bloom filter, in: IEEE Third International Conference on Convergence and Hybrid Information Technology (ICCIT), vol. 2, 2008, pp. 549–554.
[44] Y. Huang, W. Lee, Hotspot-based traceback for mobile ad hoc networks, in: Proceedings of the ACM Workshop on Wireless Security, 2005.
[45] M.S. Siddiqui, S.O. Amin, C.S. Hong, Hop-by-hop traceback in wireless sensor networks, IEEE Communications Letters 16 (2) (2012) 242–245.
[46] D.H. Kim, P.I. Hoh, B. Yoon, IP traceback methodology using markov chain and Bloom filter in 802.16e, in: Third International Conference on Convergence and Hybrid Information Technology, 2008, pp. 454–459.
[47] W. Kozma, L. Lazos, Reactive identification of misbehaviour in ad hoc networks based on random audits, in: 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, 2008, pp. 612–614.
[48] B. Liu, Y. Zhong, S. Zhang, Probabilistic isolation of malicious vehicles in pseudonym changing VANETs, in: 7th IEEE International Conference on Computer and Information Technology, 2007, pp. 967–972.
[49] C.T. Huang, LOFT: low-overhead freshness transmission in sensor networks, in: IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing, 2008, pp. 241–248.
[50] D. Jinwala, D. Patel, S. Patel, K.S. Dasgupta, Replay protection at the link layer security in wireless sensor networks, in: IEEE WRI World Congress on Computer Science and Information Engineering, vol. 1, 2009, pp. 160–165.
[51] W. Znaidi, M. Minier, S. Ubeda, Hierarchical node replication attacks detection in wireless sensors networks, in: IEEE 20th International Symposium on Personal, Indoor and Mobile Radio Communications, 2009, pp. 82–86.
[52] B. Tong, S. Panchapakesan, W. Zhang, A three-tier framework for intruder information sharing in sensor networks, in: 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, 2008, pp. 451–459.
[53] M. Zhang, V. Khanapure, S. Chen, X. Xiao, Memory efficient protocols for detecting node replication attacks in wireless sensor networks, in: 17th IEEE International Conference on Network Protocols (ICNP), 2009, pp. 284–293.
[54] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, J. Lockwood, Deep packet inspection using parallel bloom filters, IEEE Micro 24 (1) (2004) 52–61.
[55] M.H. Lee, Y.H. Choi, A fault-tolerant bloom filter for deep packet inspection, in: 13th Pacific Rim International Symposium on Dependable Computing (PRDC), 2007, pp. 389–396.
[56] J.W. Lockwood, N. Naufel, J.S. Turner, D.E. Taylor, Reprogrammable network packet processing on the field programmable port extender (FPX), in: ACM International Symposium on Field Programmable Gate Arrays (FPGA), 2001, pp. 87–93.
[57] M. Attig, S. Dharmapurikar, J.L. Lockwood, Implementation results of bloom filters for string matching, in: Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 2004, pp. 322–323.
[58] M. Ye, K. Xu, J. Wu, Y. Cui, A high performance and scalable packet pattern-matching architecture, in: International Conference on Information Networking (ICOIN), 2008, pp. 1–5.
[59] O. Erdogan, P. Cao, Hash-AV: fast virus signature scanning by cache-resident filters, in: IEEE Global Telecommunications Conference (GLOBECOM), vol. 3, 2005.
[60] N.S. Artan, H.J. Chao, Multi-packet signature detection using prefix Bloom filters, in: IEEE Global Telecommunications Conference (GLOBECOM), vol. 3, 2005.
[61] S. Dharmapurikar, J. Lockwood, Fast and scalable pattern matching for network intrusion detection systems, IEEE Journal on Selected Areas in Communications 24 (10) (2006) 1781–1792.
[62] N.S. Artan, K. Sinkar, J. Patel, H.J. Chao, Aggregated Bloom filters for intrusion detection and prevention hardware, in: IEEE Global Telecommunications Conference (GLOBECOM), 2007, pp. 349–354.
[63] M. Nourani, P. Katta, Bloom filter accelerator for string matching, in: Proceedings of 16th International Conference on Computer Communications and Networks (ICCCN), 2007, pp. 185–190.
[64] K.S.P. Arun, Flow-aware cross packet inspection using Bloom filters for high speed data-path content matching, in: IEEE International Advance Computing Conference (IACC), 2009, pp. 1230–1234.
[65] S.P.A. Kumar, High-speed signature matching in network interface device using bloom filters, International Journal of Recent Trends in Engineering 1 (1) (2009) 264–268.
[66] K. Huang, D. Zhang, A byte-filtered string matching algorithm for fast deep packet inspection, in: IEEE The 9th International Conference for Young Computer Scientists (ICYCS), 2008, pp. 2073–2078.
[67] P.C. Lin, Y.D. Lin, Y.C. Lai, Y.J. Zheng, T.H. Lee, Realizing a sub-linear time string-matching algorithm with a hardware accelerator using Bloom filters, IEEE Transactions on Very Large Scale Integration Systems 17 (8) (2009) 1008–1020.
[68] X. Kefu, Q. Deyu, Q. Zhengping, Z. Weiping, Fast dynamic pattern matching for deep packet inspection, in: IEEE International Conference on Networking, Sensing and Control (ICNSC), 2008, pp. 802–807.
[69] J.H. Gidansky, D. Stefan, I. Dalal, FPGA-based SoC for real-time network intrusion detection using counting Bloom filters, in: IEEE SOUTHEASTCON, 2009, pp. 452–458.
[70] S. Dharmapurikar, P. Krishnamurthy, D.E. Taylor, Longest prefix matching using Bloom filters, IEEE/ACM Transactions on Networking 14 (2) (2006) 397–409.
[71] Y.Z. Li, Memory efficient parallel Bloom filters for string matching, in: International Conference on Networks Security, Wireless Communications and Trusted Computing (NSWCTC), vol. 1, 2009, pp. 485–488.
[72] G. Antichi, D. Ficara, S. Giordano, G. Procissi, F. Vitucci, Counting Bloom filters for pattern matching and anti-evasion at the wire speed, IEEE Network 23 (1) (2009) 30–35.
[73] P. Lin, F. Wang, W. Tan, H. Deng, Enhancing dynamic packet filtering technique with d-left counting Bloom filter algorithm, in: IEEE Second International Conference on Intelligent Networks and Intelligent Systems (ICINIS), 2009, pp. 530–533.
[74] J. Ho, G. Lemieux, PERG: a scalable FPGA-based pattern-matching engine with consolidated Bloomier filters, in: IEEE/FPT International Conference on ICECE Technology, 2008, pp. 73–80.
[75] ClamAV, Clam Anti-virus Signature Database. <http://www.clamav.net>.
[76] N.D.A. Tuan, B.T. Hieu, T.N. Thinh, High performance pattern matching using Bloomier filter, in: International Conference on Electrical Engineering/Electronics Computer Telecommunications and Information Technology, ECTI-CON, 2010, pp. 870–874.
[77] H. Song, J. Lockwood, Multi-pattern signature matching for hardware network intrusion detection systems, in: IEEE Global Telecommunications Conference (GLOBECOM), vol. 3, 2005, pp. 5–9.
[78] Z. Shenghua, Q. Zheng, Z. Yuan, P. Xiaolan, A cascade hash design of Bloom filter for signature detection, in: IEEE International Forum on Information Technology and Applications (IFITA), vol. 2, 2009, pp. 559–562.
[79] Z. Chen, C. Lin, J. Ni, D.H. Ruan, B. Zheng, Z.X. Tan, Y.X. Jiang, X.H. Peng, A. Luo, B. Zhu, Y. Yue, Y. Wang, P. Ungsunan, F.Y. Ren,

Antiworm NPU-based prallel for TCP/IP content processing Bloom filters in giga-ethernet LAN, in: IEEE International Conference on Communications, (ICC), 2006.
[80] Y. Chen, A. Kumar, J. Xu, A new design of Bloom filter for packet inspection speedup, in: IEEE Global Telecommunications Conference (GLOBECOM), 2007, pp. 1–5.
[81] T. Kocak, I. Kaya, Low-power Bloom filter architecture for deep packet inspection, IEEE Communications Letters 10 (3) (2006) 210–212.
[82] I. Kaya, T. Kocak, A low power lookup technique for multi-hashing network applications, in: Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, 2006, pp. 179–185.
[83] A.C. Snoeren, C. Partridge, L.A. Sanchez, C.E. Jones, F. Tchakountio, B. Schwartz, S.T. Kent, W.T. Strayers, Single-packet IP traceback, IEEE/ACM Transactions on Networking 10 (6) (2002) 721–734.
[84] W.T. Strayer, C.E. Jones, F. Tchakountio, R.R. Hain, SPIE-IPv6: single IPv6 packet traceback, in: Proceedings of 29th IEEE Local Computer Networks Conference (LCN), 2004, pp. 118–125.
[85] A.C. Snoeren, C. Partridge, L.A. Sanchez, C.E. Jones, F. Tchakountio, S.T. Kent, W.T. Strayer, Hash-based IP traceback, in: Proceedings of the ACM Special Interest Group on Data Communications (SIGCOMM), 2001, pp. 3–14.
[86] T.H. Lee, W.K. Wu, T.Y.W. Huang, Scalable packet digesting schemes for IP traceback, in: Proceedings of IEEE International Conference on Communications, vol. 2, 2004, pp. 1008–1013.
[87] 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, pp. 115–129.
[88] K. Shanmugasundaram, H. Bronnimann, N. Memon, Payload attribution via hierarchical Bloom filters, in: Proceedings of 11st ACM Conference on Computer and Communications Security, 2004, pp. 31–41.
[89] L. Zhang, Y. Guan, TOPO: a topology-aware single packet attack traceback scheme, in: IEEE Securecomm and Workshops (SECCOMW), 2006, pp. 1–10.
[90] E. Hilgenstieler, E.P. Duarte, G. Mansfield-Keeni, N. Shiratori, Improving the precision and efficiency of log-based IP packet traceback, in: IEEE Global Telecommunications Conference (GLOBECOM), 2007, pp. 1823–1827.
[91] H. Tang, C. Xu, X. Luo, J. OuYang, Traceback-based Bloom filter IPS in defending SYN flooding attack, in: IEEE 5th International Conference on Wireless Communications, Networking and Mobile Computing (WiCom), 2009, pp. 1–6.
[92] H. Takurou, K. Matsuura, H. Imai, IP traceback by packet marking method with Bloom filters, in: 41th Annual IEEE International Carnahan Conference on Security Technology, CCST, 2007, pp. 255–263.
[93] Z. Zhou, B. Qian, X. Tian, D. Xie, Fast traceback against large-scale DDoS attack in high-speed internet, in: IEEE International Conference on Computational Intelligence and Software Engineering, (CISE), 2009, pp. 1–7.
[94] R.P. Laufer, P.B. Velloso, D. de O. Cunha, I.M. Moraes, M.D.D. Bicudo, M.D.D. Moreira, O.C.M.B. Duarte, Towards stateless single-packet IP traceback, in: Proceedings of 32th IEEE Conference on Local Computer Networks, 2007, pp. 548–555.
[95] D. Siradjev, L. Yunusov, Y.T. Kim, Security management with scalable distributed IP traceback, in: IFIP/IEEE International Symposium on Integrated Network Management, 2009, pp. 598–605.
[96] J. Yan, P.L. Cho, Enhancing collaborative spam detection with Bloom filters, in: 22th Annual Computer Security Application Conference (SCSAC), 2006, pp. 414–428.
[97] DCC, Distributed Checksum Clearinghouse, 2007. <http://www.rhyolite.com/anti-spam/dcc>.
[98] Razor, 2007. <http://razor.sourceforge.net>.
[99] Z. Zhong, K. Li, Speedup statistical spam filter by approximation, IEEE Transactions on Computers 60 (1) (2010) 120–134.
[100] M. Takesue, Personalized filtering of polymorphic e-mail spam, in: IEEE Third International Conference on Emerging Security Information, Systems and Technologies (SECURWARE), 2009, pp. 249–254.
[101] A.J. Aviv, M.E. Locasto, S. Potter, A.D. Keromytis, SSARES: secure searchable automated remote email storage, in: IEEE Twenty-Third Annual Computer Security Applications Conference (ACSAC), 2007, pp. 129–139.
[102] E.Y.K. Chan, H.W. Chan, K.M. Chan, V.P.S. Chan, S.T. Chanson, M.M.H. Cheung, C.F. Chong, K.P. Chow, A.K.T. Hui, L.C.K. Hui, L.C.K. Lam, W.C. Lau, K.K.H. Pun, A.Y.F. Tsang, W.W. Tsang, S.C.W. Tso, D.-Y. Yeung, K.Y. Yu, IDR: an intrusion detection router for defending against distributed denial-of-service (DDoS) attacks, in: 7th International Symposium on Parallel Architectures, Algorithms and Networks, 2004, pp. 581–586.
[103] S.H. Shim, K.M. Yoo, K.E. Han, C.K. Kang, W.H. So, J.T. Song, Y.C. Kim, Destination address monitoring scheme for detecting DDoS attack in centralized control network, in: IEEE Asia-Pacific Conference on Communications (APCC), 2006, pp. 1–5.
[104] D. Peng, G. Chang, R. Guo, Y. Tang, Research on DDoS filtering algorithm based on Bloom filter white list, in: International Conference on Multi Media and Information Technology (MMIT), 2008, pp. 291–297.
[105] C.E. Rothenberg, P. Jokela, P. Nikander, M. Sarela, J. Ylitalo, Self-routing denial-of-service resistant capabilities using in-packet Bloom filters, in: European Conference on Computer Network Defense (EC2ND), 2009, pp. 46–51.
[106] P. Du, A. Nakao, DDoS defense deployment with network egress and ingress filtering, in: IEEE International Conference on Communications, ICC, 2010, pp. 1–6.
[107] C. Sun, B. Liu, L. Shi, Efficient and low-cost hardware defense against DNS amplification attacks, in: IEEE Global Telecommunications Conference (GLOBECOM), 2008, pp. 1–5.
[108] B. Whitehead, C.H. Lung, P. Rabinovitch, A TCP connection establishment filter: symmetric connection detection, in: IEEE International Conference on Communications (ICC), 2007, pp. 247–253.
[109] D. Nashat, X. Jiang, S. Horiguchi, Detecting SYN flooding agents under any type of IP spoofing, in: IEEE International Conference on e-Business Engineering, 2008, pp. 499–505.
[110] C. Sun, J. Fan, B. Liu, A robust scheme to detect SYN flooding attacks, in: International Conference on Communications and Networking in China, 2007, pp. 397–401.
[111] C. Sun, C. Hu, Y. Tang, B. Liu, More accurate and fast SYN flood detection, in: Proceedings of 18th IEEE International Conference on Computer Communications and Networks (ICCCN), 2009, pp. 1–6.
[112] A. Patcha, J.M. Park, Detecting denial-of-service attacks with incomplete audit data, in: Proceedings of 14th IEEE International Conference on Computer Communications and Networks (ICCCN), 2005, pp. 263–268.
[113] J. Fan, D. Wu, K. Lu, A. Nucci, Design of Bloom filter array for network anomaly detection, in: IEEE Global Telecommunications Conference (GLOBECOM), 2006, pp. 1–5.

Shahabeddin Geravand received the B.S. degree in Computer engineering – software from Islamic Azad University of Arak, Arak, Iran in 2008. In May 2011, he received the M.Sc. degrees in Computer engineering – software from Islamic Azad University, Arak, Iran. His research interests include network security, and database.

Mahmood Ahmadi received the B.S. degree in Computer engineering from Isfahan University, Isfahan, Iran in 1995. He received the M.Sc. degrees in Computer architecture and engineering from Tehran Polytechnique University, Tehran, Iran in 1998. From 1999 to 2005, he was a faculty member at Razi University in Kermanshah in Iran. In October 2005, he joined the Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS), Delft University of Technology, Delft, The Netherlands, as a full-time Ph.D. student. He got his PhD in May 2010. His research interests include Computer architecture, network processing, signal processing, and reconfigurable computing. He is currently working as an assistant professor in the Department of Computer Engineering at the Razi University of Kermanshah. He is a member of the IEEE, and HIPEAC.