Future Generation Computer Systems 29 (2013) 1838–1850

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

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

A confidence-based filtering method for DDoS attack defense in cloud environment

Wanchun Dou[a,*], Qi Chen[a], Jinjun Chen[b]

[a] State Key Laboratory for Novel Software Technology, Nanjing University Nanjing, 210093, PR China

[b] School of System, Management and Leadership, University of Technology, Sydney, Australia

ARTICLE INFO

Article history:
Received 24 August 2012
Received in revised form
4 December 2012
Accepted 8 December 2012
Available online 27 December 2012

Keywords:
Distributed denial-of-service attack
Filtering
Confidence
Correlation characteristic
Network security
Cloud environment

ABSTRACT

Distributed Denial-of-Service attack (DDoS) is a major threat for cloud environment. Traditional defending approaches cannot be easily applied in cloud security due to their relatively low efficiency, large storage, to name a few. In view of this challenge, a Confidence-Based Filtering method, named CBF, is investigated for cloud computing environment, in this paper. Concretely speaking, the method is deployed by two periods, i.e., non-attack period and attack period. More specially, legitimate packets are collected in the non-attack period, for extracting attribute pairs to generate a nominal profile. With the nominal profile, the CBF method is promoted by calculating the score of a particular packet in the attack period, to determine whether to discard it or not. At last, extensive simulations are conducted to evaluate the feasibility of the CBF method. The result shows that CBF has a high scoring speed, a small storage requirement, and an acceptable filtering accuracy. It specifically satisfies the real-time filtering requirements in cloud environment.

1. Introduction

1.1. Current status of related research

Cloud computing is a long-held dream of computing as a utility. As discussed in [1], it has the potential to transform a large part of the IT industry, making software even more attractive as a service and shaping the way IT hardware is designed and purchased. Nowadays, it is evolving as a key computing platform for sharing resources including infrastructure resources, software resources, application resources and business processes [2]. However, with the large amount of resources online, these cloud systems are facing severe security problems.

*D*istributed *D*enial-of-*S*ervice (DDoS) attack can be considered as a major threat to cloud computing. The attackers often compromise vulnerable hosts, called *zombies*, on the network and install attack tools on them. These zombies together form a *botnet* and will generate large amount of distributed attack packets targeting at the victims under the control of the attackers. This attack will block the legitimate access to the servers, exhaust their resources such as network bandwidth, computing power and even lead to great financial losses as shown in [3].

In recent years, many researches on DDoS defense have been carried out and many successful schemes have been put forward. There are approximately three major branches of the research: attack detection [4–6], attack filtering [7–12], and attack traceback [13–15].

As mentioned in [7], the branch of attack filtering can be roughly categorized into three areas based on the point of protection: source-initiated, path-based and victim-initiated. The method proposed in this article is in victim-initiated area, which filters incoming attack packets from victim side. In this area of research, a number of brilliant approaches have already been proposed.

PacketScore [7] generates value distributions of some attributes in the TCP and IP headers, and then uses Bayes' Theorem to score packets. PacketScore has a pretty high filtering accuracy and it is also easily deployed. But since its scoring and discarding are related to attack intensity, it is not suitable for handling large amounts of attack traffic. Also it has some costly operations in scoring, which leads to low processing efficiency in real-time filtering.

ALPi [8] is an improvement of PacketScore. Two schemes, LB and AV, which use leaky buckets and value variances of attributes respectively are proposed and are evaluated by comparison with PacketScore. *H*op-*C*ount *F*iltering (HCF) [9] uses the relationship of source IP address and TTL value to carry out filtering. After building an IP to hop-count mapping, it can detect and discard spoofed IP packets with about 90% accuracy. It is effective and easily deployed but it is vulnerable to distributed attacks because of its assumption about spoofed IP traffic.

Our method aims at mining the correlation characteristics, which refer to some simultaneously-appeared characteristics in

[*] Corresponding author.
E-mail address: douwc@nju.edu.cn (W. Dou).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.12.011

the legitimate packets. [16,17] use the document popularity and user browsing behaviors to detect attack packets, which reflect some correlation characteristics between packets in a flow. But these characteristics are mainly in the application layer, making these methods mostly effective for application layer DDoS.

1.2. Motivation

Considering more and more resources are being shared in cloud platforms, especially in an elastic cloud environment which could nearly provide unlimited capabilities [18], the effect of DDoS attacks can be not only much more powerful and influential, but also in a much wider range. In view of this challenge, this paper aims at proposing a method for cloud security, which aims to be much quicker in responding, easier to be widely deployed and more powerful in ability than before.

1.3. The organization of the paper

This paper is organized as follows: In Section 2, the concept of correlation and its unique characteristic is introduced, which forms the basis of our method. In Section 3, we first introduce some preliminary knowledge, and then give a scenario of our method, *C*onfidence-*B*ased *F*iltering. In the next two sections, we focus on the details of the behaviors of our method in two periods, the non-attack and the attack period respectively. In Section 6, the performance of our method under different types of DDoS attacks are evaluated based on real world traffic. Section 7 discusses some important issues about the ability of the method, and at the end Section 8 gives a brief conclusion and the direction of future work.

2. Correlation characteristic

2.1. Definition of correlation characteristic

In order to discriminate attack packets from legitimate ones, the method proposed in this paper utilizes correlation characteristics. Our method defines the concept of correlation as follows,

Definition 1 (*Correlation*)**.** Correlation is the situation that some interior events in the packet flows take place at the same time.

Take the correlation in user browsing behaviors as an example. For NBA fans that live in Los Angeles, the majority of them tend to love the team Los Angeles Lakers. So when they log onto the website of ESPN, the servers of ESPN will receive more packets which request the Lakers webpage and have the IP addresses from the area around Los Angeles as the same time. So this is a correlation between webpage requests and the source IP addresses.

With the definition of correlation, we can define correlation characteristic as follows,

Definition 2 (*Correlation Characteristic*)**.** Correlation characteristic is the distinguishing feature of a correlation situation.

In our method, the correlation situations will be quantized to values, so the corresponding correlation characteristics are their value distributions.

2.2. Utilizing correlation characteristic

Like user browsing behaviors mentioned above, there are a large amount of correlation characteristics and some of them can be very complicated. These kinds of correlation characteristics are different for different user, server and application, which will be quite hard for attackers to notice and mimic. Considering that, using this kind of characteristics to judge the legitimacy of packets can be feasible, which is the main idea of this paper.

More specifically, our method focuses the probe on transport and network layers. The correlation characteristics that we mine in these two layers are the co-appearances between attributes in IP header and TCP header. These attribute pair characteristics are distinctive because certain characteristics of the operating system, network structure and even hobbies of users can affect the values of these attributes, and thus make some attribute pairs related. In [9], the hop-count filtering constructs an IP2HC table which maps source IP addresses to TTL values, and filters attack packets by checking the validation of TTL according to the source IP address. It can be seen that the key point of its success is utilizing the correlation characteristics between TTL and source IP address. In view of this, it is reasonable to generalize this idea to all correlation characteristics between attributes in IP header and TCP header.

So the basic assumption of this idea is that there are indeed some unique correlation characteristics for all attribute pairs in IP header and TCP header in legitimate packet flows. To validate this assumption, we extract consecutive 10^5 packets from samplepoint-B in the MAWI Traffic Archive [19]. We count the occurrences of the values of 15 attribute pairs in IP header and TCP header, and study their value distributions. In Fig. 1, we show four representative ones among them, in (a)–(d) respectively. All of the attribute values we select are 16-bits (for 32-bits attributes like IP addresses, we only use the first 16-bits of them), and the x-axis in Fig. 1 is the combined values of the attribute pairs, which are 32-bits. The y-axis is the number of appearances of the attribute value pairs.

As shown in Fig. 1, all 4 types of value distribution are much different from common ones such as normal distribution, uniform distribution, Poisson distribution, etc. They tend to have peaks in some certain attribute value pairs, and have nearly no occurrences in the other value pairs. This indicates that for each attribute pair, some values appear much more frequently than the other ones, and this unusual distribution is unlikely to be mimicked if the attackers do not own as much packet flow information as the victim.

Next we will test whether these correlation characteristics are unique for different end systems. If they are unique enough, the attackers are unlikely to build a universal attack flow which can break down all end systems which use our method. We extract consecutive 10^5 packets from another samplepoint in the MAWI Traffic Archive [19], samplepoint-A, to do the same experiment as the one with samplepoint-B described above. In (e)–(h) of Fig. 1, we present value distributions of the same attribute pairs as the ones in (a)–(d).

From comparisons between (a) and (e), between (b) and (f), and between (c) and (g), the value distributions of these three attribute pairs are quite different for samplepoint-A and samplepoint-B. This indicates that most value distributions of attribute pairs can be viewed as unique characteristics for each end system. The only exception happens in the comparison between (d) and (h), which shows that the value distributions of source IP address and destination port number pair are similar to each other for samplepoint-A and samplepoint-B. This reminds us that value distributions for few attribute pair may not be unique. Considering that, all attribute pairs should be taken into consideration and be analyzed in a comprehensive way if we want to mine them as unique characteristics for end systems.

2.3. Conclusion

There are some unique correlation characteristics between attributes in IP header and TCP header in legitimate packet flows for each end system. In view of this, if attribute pair value distributions of the attack flows are not similar to those of victim's,

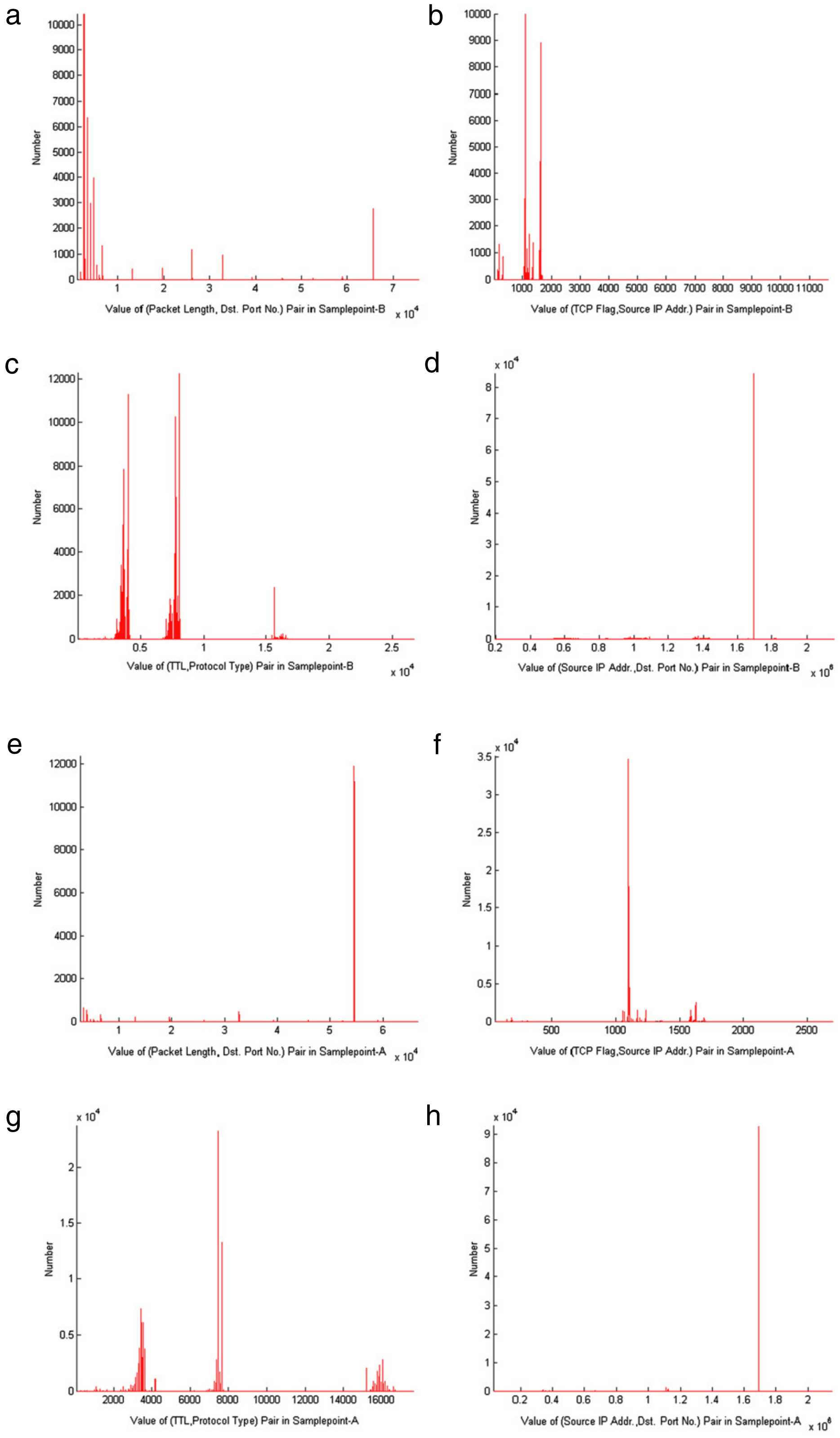


Fig. 1. Value distributions of 4 types of attribute pairs in IP header and TCP header from samplepoint A and B. (a), (e) are the value distribution of the total packet length and destination port number pair from samplepoint A and B respectively; (b), (f) are the value distribution of TCP flag and source IP address pair from samplepoint A and B respectively; (c), (g) are the value distribution of TTL and protocol type pair from samplepoint A and B respectively; (d), (h) are the value distribution of source IP address and destination port number pair from samplepoint A and B respectively.

we can utilize these correlation characteristics in our method to distinguish legitimate packets from attacker ones.

Fortunately, to the best of our knowledge, most DDoS attacks do not notice and mimic the value distributions of attributes in IP and TCP headers when generating packets. In fact, they do not have any chance to mimic most of them, since they need to fill many attribute values to meet their own attack requirements. For example, as shown in [3], attackers need to fix the protocol type

Table 1
Key terms appeared in this paper.

Terms	Description
n	The number of the attributes under consideration in the method
A_i	The i-th attribute in the packet, $(1 \le i \le n)$
m_i	The number of values which attribute A_i can have
$a_{i,j}$	The j-th value of attribute A_i, $(1 \le j \le m_i)$
t	A time interval in packet flows
N_n	The total number of packets in the packet flow in one time interval t
$N(A_i = a_{i,j})$	The number of packets whose attribute A_i has value $a_{i,j}$ in this packet flow in one time interval t
$N(A_r = a_{r,x}, A_s = a_{s,y})$	The number of packets whose attribute A_r has value $a_{r,x}$, attribute A_s has value $a_{s,y}$ in this packet flow in one time interval t
p	A packet in the packet flows
$p(i)$	The value of attribute A_i in packet p

values when launching protocol-based attacks such as SYN flood, CMP flood, HTTP flood, etc. When in need of spoofing IP addresses and port numbers, they simply use random values. Considering that, the attribute pairs' value distributions in attack flows are not likely to resemble those in legitimate flows. It makes our idea feasible enough.

In the rest of this paper, we will design a feasible and effective filtering method, which can leverage the correlation characteristics of attribute pairs in IP header and TCP header to defend against DDoS Attack.

3. Confidence-based filtering method

3.1. Preliminary knowledge

3.1.1. Key terms

To help illustrate our method, some key terms used in this paper are summarized in Table 1.

3.1.2. Confidence and CBF score

In this part, we will first introduce two concepts: the one named confidence for measuring correlation characteristics, and the one named CBF score for judging the legitimacy of packets. With the concept of CBF score, we will define the CBF legitimacy of a packet.

The concept of confidence reflects how much trust we can put on a correlation characteristic between attributes. Similar to the concept of confidence defined in association rules in a data mining area [20], in this paper we define it formally as follows,

Definition 3 (*Confidence*)**.** Confidence is the frequency of appearances of attributes in the packet flows. The confidences (*Conf* for short) for single attributes and attribute pairs are calculated as (1) and (2),

Confidence for single attributes:

$$Conf(A_i = a_{i,j}) = \frac{N(A_i = a_{i,j})}{N_n}, \tag{1}$$

where $i = 1, 2, 3, \ldots, n, j = 1, 2, 3, \ldots, m_i$,

Confidence for attribute pairs:

$$Conf(A_{i_1} = a_{i_1,j_1}, A_{i_2} = a_{i_2,j_2}) = \frac{N(A_{i_1} = a_{i_1,j_1}, A_{i_2} = a_{i_2,j_2})}{N_n}, \tag{2}$$

where $i_1 = 1, 2, 3, \ldots, n, i_2 = 1, 2, 3, \ldots, n, j_1 = 1, 2, 3, \ldots, m_1, j_2 = 1, 2, 3, \ldots, m_2$.

In (1) and (2), the meanings of the variables are listed in Table 1.

Indicated by Definition 3, the more times an attribute pair appears in the legitimate packet flows, the higher the confidence value of this pair we can get. The concept of confidence is the basis of the calculation of CBF score and the whole filtering process, so we name our method Confidence-Based Filtering, CBF for short.

With confidence values of attribute value pairs, the legitimacy criterion of a packet is defined as follows,

Definition 4 (*CBF Score*)**.** CBF score for a packet is the weighted average of the confidence of the attribute value pairs in it. The CBF score for a packet p is calculated as (3):

$$Score(p) = \frac{\sum_{k=1}^{d} W(A_{k_1}, A_{k_2}) Conf(A_{k_1} = p(k_1), A_{k_2} = p(k_2))}{\sum_{k=1}^{d} W(A_{k_1}, A_{k_2})}. \tag{3}$$

In this definition, d is the total number of the attribute pairs involved in the calculation of score. A_{k_1} and A_{k_2} are two attributes in the k-th attribute pair. $W(A_{k_1}, A_{k_2})$ is the weight for the k-th attribute pair. Considering the range of each confidence value is in [0, 1], the range of $Score(p)$ is also in [0, 1]. Choosing weighted average confidence values of all attribute pairs can decrease the effect of the exception cases described in Section 2.

Thus, in order to calculate CBF scores of packets, we need to prepare the confidence of each attribute value pair in legitimate packet flow beforehand. In our method, we design a dataset for these confidence values, named *nominal profile*. The generating details of it are discussed in Section 3.

In (3), the attribute pairs which cannot be easily copied by attackers will be given a high weight. Thus, a higher score of a packet corresponds to more frequently-appeared and difficultly-copied correlation characteristics, and thus more likely to be legitimate. So we can choose a *discarding threshold* to make the judgment of filtering. In view of this, the legitimacy of the packets is defined as follows,

Definition 5 (*CBF Legitimate Packet*)**.** The legitimate packet in CBF is the one whose CBF score is above the *discarding threshold*.

So on the contrary, those packets with scores lower than the *discarding threshold* are regarded as attack ones.

3.2. Confidence-based filtering

The overall process of CBF method can be divided into two periods: non-attack period and attack period. A scenario of our method is shown in Fig. 2. Its details will be introduced in the following sections.

In the non-attack period, the main target is to generate *nominal profile*. For incoming packets, our method firstly extracts the required attribute value pairs from them. With (2) in Definition 3, the number of appearances of these value pairs will be counted and their confidence values calculated. Then these confidence values are used to update *nominal profile*.
In the attack period, most packets are not legitimate, so CBF will stop generating *nominal profile*. Like in the non-attack period, extracting the attribute value pairs from the incoming packets is the first step. With these value pairs, our method searches *nominal profile* for their confidence values in legitimate flows. Then CBF score, the filtering criterion, is calculated using (3) in Definition 4. After a packet discarding strategy is selected, CBF will judge the legitimacy of the packet based on Definition 5, and decide to let it pass or not.

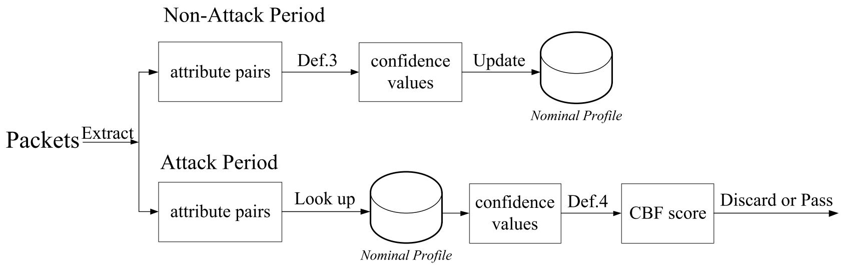


Fig. 2. Scenario of confidence-based filtering.

Table 2
Single attributes selected from IP/TCP header.

Location	Attribute	Description
IP header	Total length	The length of the datagram, measured in octets, including internet header and data.
	Time to Live (TTL)	The maximum time the datagram is allowed to remain in the internet system
	Protocol type	The next level protocol used in the data portion of the IP datagram
	Source IP address	The destination IP address (our method uses the 16-bit prefixes of it)
TCP header	Flag	Control bits that indicate different connection states or information about how a packet should be handled
	Destination port number	The destination port number

Table 3
Example of *nominal profile* with two attribute pairs.

Attribute value pair		TTL, packet size		TTL, TCP flag	
1	1	TTL = 1, packet size = 1	0.1%	TTL = 1, TCP flag = 1	0.01%
1	...	TTL = 1, packet size = ...	...	TTL = 1, TCP flag = ...	...
1	255	TTL = 1, packet size = 255	1.5%	TTL = 1, TCP flag = 255	1.2%
2	1	TTL = 2, packet size = 1	0.1%	TTL = 2, TCP flag = 1	0.05%
...	...	...	...	...	...
255	255	TTL = 255, packet size = 255	0.3%	TTL = 255, TCP flag = 255	0.08%

4. Confidence-based filtering method in the non-attack period

4.1. Nominal profile structure

In this part, we will introduce the structure of nominal profile. Firstly, we select six candidate single attributes as shown in Table 2. Then, we combine every two (not the same) of the six attributes and get 15 attribute pairs. After combination, the values of attribute pair will have 32-bit sizes since the 6 single attributes all have sizes of no more than 16-bit. Table 3 shows an example of the nominal profile structure which contains two attribute pairs (TTL, packet size) and (TTL, source IP address).

The overall construction of the *nominal profile* is divided into small time intervals, which are called windows. The size of a window can be set to fixed ones or dynamic ones. In each time interval t, our method CBF counts the number of the value appearances of these 15 attribute pairs, and then uses Definition 3 to calculate the confidence values. At the end of each time interval, the new confidence values are used to update the *nominal profile*. In order to minimize the false negative rate, the highest confidence value of an attribute value pair in the *nominal profile* is stored, which means the updating only takes place when the new confidence value is higher than the one stored in the *nominal profile*.

4.2. Profile storage saving

In order to construct the *nominal profile*, CBF calculates the confidence values of every attribute value pair and stores them in a certain data structure. However, this may incur a storage problem. The common strategy for storing them will use a 3-dimension array. The first dimension is for the attribute pair and has a length of 15. The second dimension is the value set of a certain attribute pair, which has 32-bit size. The third dimension is the confidence value dimension and the size of it depends on the precision requirement of confidence values. If we use 32-bit for the third dimension, the overall needed storage will be $15 \times 2^{32} \times 4$ bytes, which is equal to 240 GB. This amount of storage cannot be feasible in practice.

This storage problem takes place in two steps in our method: the counting step and the storing step. In the counting step, our method needs this 3-dimension array to prepare spaces for counting the attribute pair value appearances. In the storing step, we also need the same kind of 3-dimension array to store the confidence values.

The solution to this problem starts from modifying the storing style of the *nominal profile*. So we will first introduce the storage saving strategy in storing step, and then introduce the strategy in the counting step. Here, the performance evaluation of our storage saving is shown.

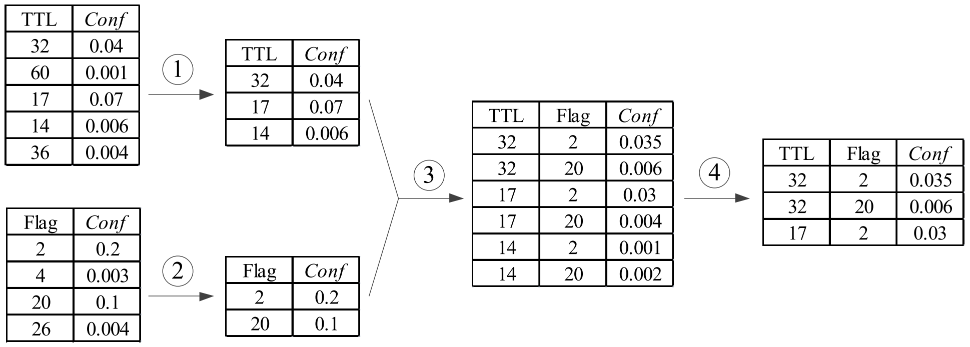
4.2.1. Storage saving in storing step

In the storing step, we can use iceberg-style profiles [21]. In this implementation, we only store the confidence values of attribute value pairs which are higher than a predetermined threshold, e.g., 0.001%. We call this threshold *minconf*, which means the minimum confidence value in the *nominal profile*.

As studied in Section 2, most of the attribute value pairs have no occurrence, which means their confidence values will be zero or very small if above zero. So if using *minconf* as the baseline, most of the attribute value pairs will not be stored, cutting the size of the needed data storage down proportionately. So with a hash function to search and store them, this storage problem in the storing step is successfully solved.

4.2.2. Storage saving in counting step

After using the iceberg-style profile, we only need to handle the attribute value pairs whose confidence values are greater than *minconf*. In view of this, we can generate confidence values of



TTL	*Conf*
32	0.04
60	0.001
17	0.07
14	0.006
36	0.004

Flag	*Conf*
2	0.2
4	0.003
20	0.1
26	0.004

TTL	*Conf*
32	0.04
17	0.07
14	0.006

Flag	*Conf*
2	0.2
20	0.1

TTL	Flag	*Conf*
32	2	0.035
32	20	0.006
17	2	0.03
17	20	0.004
14	2	0.001
14	20	0.002

TTL	Flag	*Conf*
32	2	0.035
32	20	0.006
17	2	0.03

Fig. 3. Example of generating confidence of attribute value pairs (TTL, Flag) from confidence of single attribute TTL and single attribute Flag. The *minconf* in this example is set to 0.005.

Table 4
Descriptions of the operations in Fig. 3.

No.	Description
1	Extract the TTL values whose *Conf* is greater than 0.005 (*minconf*)
2	Extract the Flag values whose *Conf* is greater than 0.005 (*minconf*)
3	Combine the values of TTL and Flag whose *Conf* is greater than 0.005 (*minconf*), then calculate the *Conf* for these combined values
4	Extract the values of attribute pair (TTL, Flag) whose *Conf* is greater than 0.005 (*minconf*)

attribute value pairs by the confidence values of single attribute values, which can solve the storage problem in counting step. This solution is based on the following observation:

Observation. If the confidence value of any single attribute value in an attribute value pair is not greater than *minconf*, the confidence value of this value pair will still not be greater than *minconf*.

Proof. For attribute value pair $(A_r = a_{r,x}, A_s = a_{s,y})$, suppose we have $Conf(A_r = a_{r,x}) \leq minconf$. According to (1) defined in Definition 3, $N(A_r = a_{r,x}) \leq N_n \cdot minconf$. Since $N(A_r = a_{r,x}, A_s = a_{s,y}) \leq N(A_r = a_{r,x})$, we have $N(A_r = a_{r,x}, A_s = a_{s,y}) \leq N_n \cdot minconf$. According to (2) defined in Definition 3, $Conf(A_r = a_{r,x}, A_s = a_{s,y}) \leq minconf$. □

So we can firstly count the number of appearances of single attribute values and calculate the confidence of them using (1) in Definition 3. Then we get the candidate attribute value pairs from the combination of only single attribute values whose confidence values are greater than *minconf*. So at the counting step, we will only prepare storage spaces for the candidate attribute value pairs instead of all possible ones. So at the last step, we select the attribute value pairs in the candidate ones whose confidence values are higher than *minconf* to update the iceberg-style profile.

Fig. 3 shows an example of this storage saving process, and Table 4 gives the descriptions of operations which are circled in the figure. In this example, we generate the values of attribute pair (TTL, Flag) whose confidence values are greater than 0.005 (*minconf*) from the confidence values of single attribute TTL and Flag. First we calculate the confidence of the TTL values and Flag values, and then extract 3 TTL values and 2 Flag values whose confidence values are greater than 0.005. Then we combine these 3 TTL values and 2 Flag values, and get 6 (TTL, Flag) candidate attribute value pairs. We calculate the confidence of these value pairs, and extract 3 of them whose confidence values are greater than 0.005. These 3 (TTL, Flag) value pairs and their confidence values are the data which we need to update the *nominal profile*.

4.2.3. Performance of the storage saving strategy

Table 5 shows the storage requirements for 3 min data in a trace recorded in WAWI Traffic Archive [19]. We set the window

Table 5
Profile storage requirements for different *minconf* values at storing and counting period.

minconf	Storing period		Counting period	
	Number of confidence values	Size of confidence values (kB)	Average number of counting spaces	Size of counting spaces (kB)
0.01	177	0.691	175.393	0.685
0.001	2 213	8.645	1138.607	4.448
0.0005	5 242	20.477	2100.714	8.206
0.0001	54 120	211.406	9080.893	35.469
0.00005	210 900	823.828	15 978.429	62.416

Step 1. Count the number of appearances of single attribute values and then calculate the confidence of them;
Step 2. Select the single attribute values with confidence higher than *minconf* to generate the candidate attribute value pairs;
Step 3. Scan the packet flow again to count the occurrences of the candidate attribute value pairs and calculate their confidence;
Step 4. Use the confidence values which are higher than *minconf* to update the nominal profile;

Fig. 4. Details of CBF in one window in the non-attack period.

size to 5 s and use 32-bit to store each confidence value and each counting space. For measuring storing period storage, we count the number of confidence values which are actually stored in iceberg-style profile after processing all 3 min data. For counting period, we calculate the average number of needed counting spaces for candidate attribute value pairs in each window. The result in the table shows that even using extremely low *minconf* like 0.00005, the storage usage at storing period and counting period will not exceed 1 MB, which is much less than 240 GB. And the storage requirement of a proper *minconf* like 0.001 is around 8 kB at storing period and 4.5 kB at counting period, which is feasible in most cases.

This sharp cutting down in storage also indicates that the frequently-appeared attribute value pairs only make up a small share of all possible value pairs, which validates our observations in Section 2. As shown in Section 6, a *minconf* around 0.0005 can be effective in filtering. So the core storage size for CBF is only about 20 kB, which makes it easily deployed in cloud platforms.

4.3. Non-attack period process details

Based on the solution of storage saving, a more specified process in the non-attack period during one time window is described in Fig. 4.

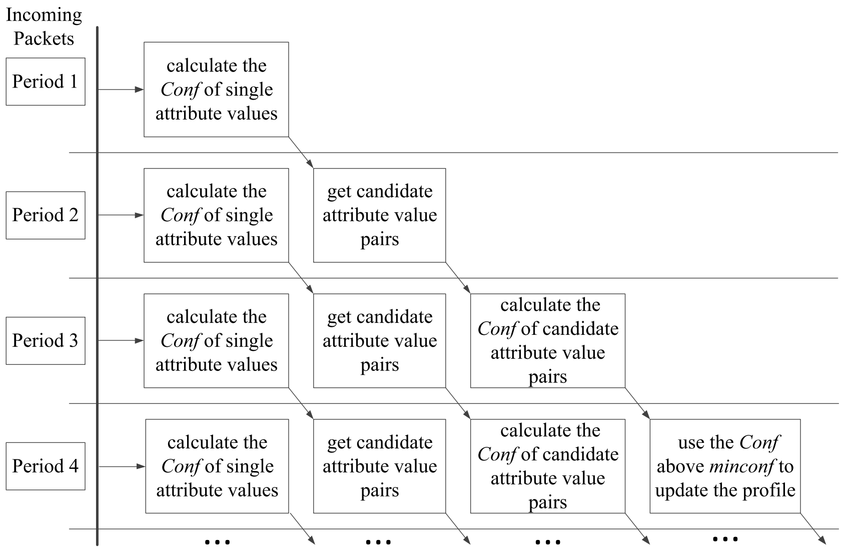

Incoming Packets
Period 1
Period 2
Period 3
Period 4
calculate the Conf of single attribute values
get candidate attribute value pairs
calculate the Conf of candidate attribute value pairs
use the Conf above minconf to update the profile
...

Fig. 5. Pipeline implementation time line for CBF in the non-attack period.

This is a 4-pass process and it can be largely accelerated if being carried out in parallel. As shown in Fig. 5, the single attribute value counting, the candidate attribute value pair generating, the second time scanning and updating profile can be put in a pipeline implementation, which will make CBF much faster, and more suitable for real-time filtering and cloud computing.

5. Confidence-based filtering method in the attack period

5.1. Calculating CBF score

Indicated in Fig. 2, in attack period CBF will firstly look up the *nominal profile* for the confidence values corresponding to the attribute value pairs in the current packets and then calculate the scores for them. In most cases, the confidence values of frequently-appeared attribute value pairs will be found in *nominal profile* successfully. But considering that we use an iceberg-style profile, the confidence of some rarely-appeared attribute value pairs will be absent. In this case, we will use *minconf* value instead when these confidence values are required in score calculation.

The adjustments of the attribute pair weights will take into consideration the unique characteristics of the operating system, the network structure and many other elements. The general idea is to make more outstanding the correlation characteristics which are less possible to be copied by attackers and more related to the inherent features of the server. For example, when under a denial-of-service attack, the source IP address in a packet is spoofed in most cases. So we can give the attribute pairs including source IP address a higher weight. On the other hand, we can give the attribute pairs including protocol type or TCP flag a lower weight because the ranges of their values are limited, thus it is easy for attackers to guess.

Fig. 6 gives an example of the scoring process, and Table 6 gives the descriptions of operations which are circled in the figure. In this example, we assume that only 3 single attributes are involved in CBF filtering, which are TTL, IP protocol and TCP flag respectively. The scoring process starts from looking up the confidence of attribute value pairs in *nominal profile*. Because of the iceberg-style storing, we cannot find the confidence of the value pair in which TTL is 30 and IP protocol is 6. So we use *minconf* to represent its possible confidence value. Then a weighted average calculation is carried out with these confidence values to generate the CBF score

Table 6
Descriptions of the operations in Fig. 6.

No.	Description
1	Find entry (TTL = 30, IP protocol = 6) in *nominal profile*
2	Find entry (IP protocol = 6, TCP flag = 2) in *nominal profile*
3	Find entry (TTL = 30, TCP flag = 2) in *nominal profile*
4, 6	Find the confidence value successfully
5	Cannot find the confidence value, use *minconf* instead
7	Weighted average the confidence values from 4, 5, 6

for this incoming packet. If the weights for (TTL, IP protocol), (IP protocol, TCP flag) and (TTL, TCP flag) are 5, 1 and 3, the CBF score for the packet in the example is given by

$$\frac{(5 \times 0.0005 + 1 \times 0.1 + 3 \times 0.09)}{(5 + 1 + 3)} = 0.0414.$$

The scoring part of CBF only requires a few looking-ups in *nominal profile* and some arithmetic operations. The asymptotic time complexity of CBF at this period is in $O(1)$, so it will be fast enough even if large amount of packets burst in when under denial-of-service attack.

5.2. Discarding strategy

After the CBF scores of packets are generated, we will use them to distinguish attack packets from legitimate ones. According to Definition 5, CBF will only accept the packets with scores greater than the *discarding threshold*. Thus for the example in Fig. 6, if the *discarding threshold* is 0.03, the packet will be judged legitimate. On the other hand, if it is 0.05, the packet will be an attack one.

The *discarding threshold* can be fixed based on the CBF score distribution of legitimate packets. According to Definition 4, the CBF score is independent from the utilization of the victim, so the fixed *discarding threshold* is feasible if the distribution of the scores is known. And the processing speed will be very high with a fixed *discarding threshold*.

Also dynamic *discarding threshold* can be adopted. Like the load-shedding algorithm used in [22], we first use current utilization of the victim and the maximum utilization to generate the amount (Φ) of suspicious traffic. As shown in Fig. 7, we can then generate the cumulative distribution function (CDF) of the scores in current

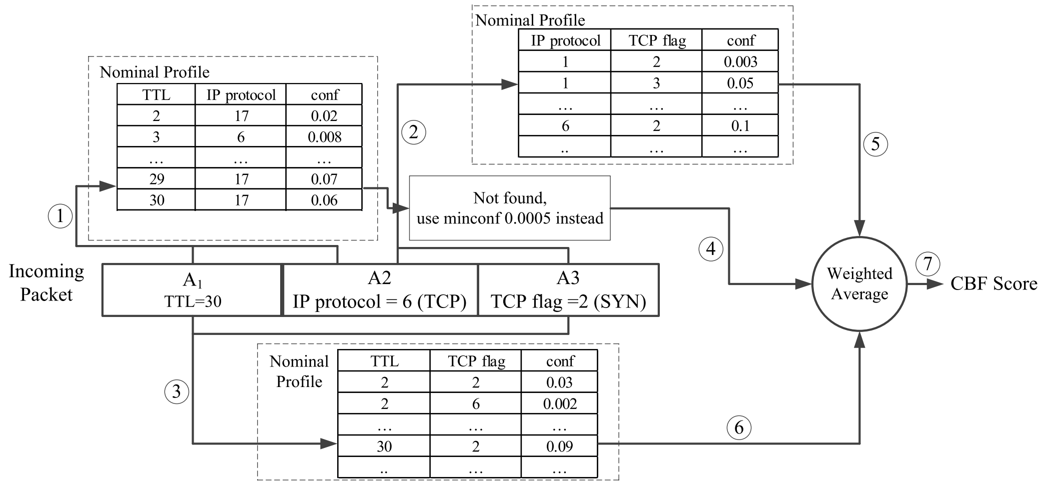


Fig. 6. Example of the scoring process in CBF. The circled numbers are the operations in scoring process, which are described in Table 6.

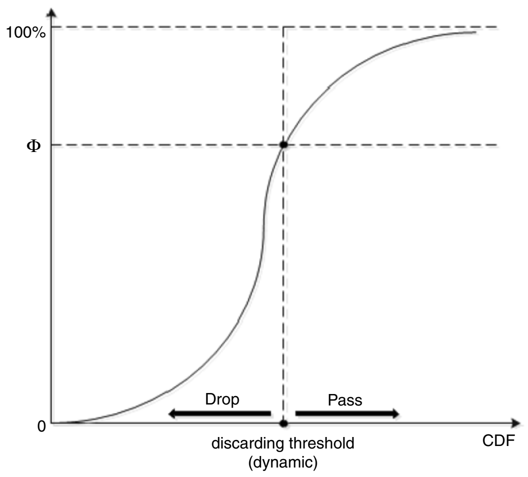


Fig. 7. Decide the dynamic *discarding threshold* using Φ.

Table 7
The hardware and software environments of our experiments.

Hardware environment	Processor	Intel Core 2 Duo processor (2.26 GHz)
	Memory	2 GB
Software environment Environment	Attack simulation program Defense algorithms (CBF and PacketScore)	Written in C++ Written in C++

Table 8
Parameter values of CBF and PacketScore used in the experiment.

Parameter	Value	
	CBF	PacketScore
Window size (s)	5	5
minconf	0.005	0.01
Selection of single attributes	The 6 attributes in Table 2	The 6 attributes in Table 2
Discarding threshold selection strategy	Fixed	Dynamic

time window and decide the *discarding threshold* using Φ. So the packets whose CBF scores are below this *discarding threshold* are the Φ amount most suspicious ones in the current traffic, which we need to discard. However, this dynamic *discarding threshold* calculation may incur the additional scores counting and CDF computing, which will be slightly slower than a fixed one.

6. Performance evaluation

In this section, we will use real world statistics to test the filtering method CBF. For the input of our method, we adopt the data in the MAWI Traffic Archive [19], which contains many online traffic data collections. Worth noting is that our method utilizes the resources in transport and network layers, and the basic architectures in these layers are almost the same in cloud and normal network environments. So the result of our experiment in the normal network environment mentioned above is sufficient to show our performance in cloud environment.

The hardware and software environment of the evaluation is listed in Table 7. In this section, we will firstly introduce the simulation conditions including the data source, the parameter selection for the method and different attack types. The result is shown and analyzed by taking into consideration the comparison with PacketScore [7].

6.1. Simulation conditions

6.1.1. Data

We select the data from samplepoint-B in MAWI Working Group Traffic Archive [19]. The part of data used in this section is collected from 14:00:00 to 14:15:00 on Jan 1, 2006. There are about 6 587 564 packets (2395.28 MB) contained in this data set and the average rate is 22.33 Mbps.

Every second, the data set has around 6000–7000 packets.

6.1.2. Parameters

The values of the common parameters of CBF and PacketScore which are used in our experiment are listed in Table 8.

Both of the methods choose the same six single attributes shown in Table 2. While PacketScore use the six single attributes directly, for CBF we need to combine them to get 15 attribute pairs, as described in Section 4.
For CBF method, the value of *minconf* is set to 0.005. Under this circumstance, the storage will be around 20 kB at storing period and 8 kB at counting period, which is affordable in normal servers. The window size is set to be 5, and our method spends around 0.4 s to process data during each time window. We believe this time can be minimized sharply after using pipeline implementations shown in Fig. 5 and optimizations of the algorithm codes.

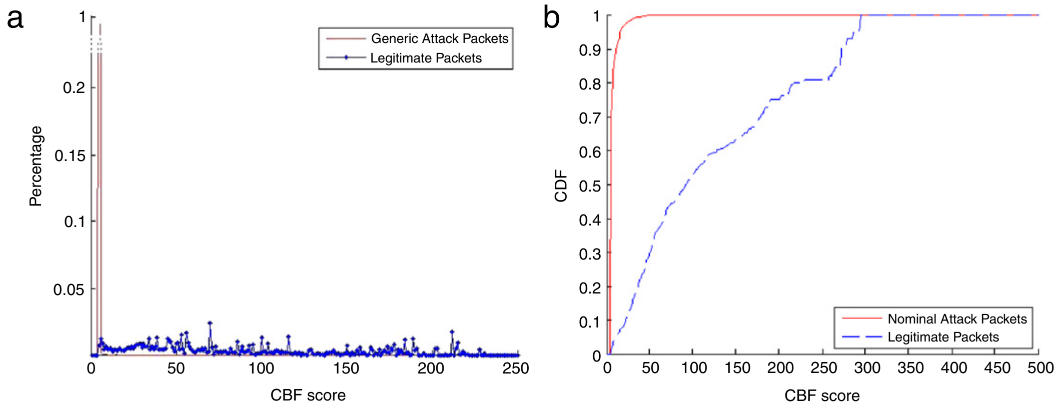


Fig. 8. CBF score distribution of attack flow and legitimate flow.

The weights in CBF score calculation are set higher in the attribute pairs containing source IP address, TCP server port number or TTL value, and set lower in those only with TCP flag, IP protocol type and packet size. For the fast response in the attack period, fixed *discarding threshold* is adopted.

For PacketScore method, the window size is also set to be 5, and the threshold for iceberg-style profile is 0.01. In our implementation, the discarding percentage is selected dynamically by a load-shedding algorithm [22], and CDF is used to calculate the *discarding threshold* of the packet score.

6.1.3. Attack types

In this evaluation, we simulate the following types of attacks:

(a) *Generic attack*.
All attributes in the attack packets are selected randomly in their allowable ranges.

(b) *TCP-SYN Flood attack*.
The TCP SYN flag is set in each attack packets and the packet lengths of them are set to be 40. Other attributes are selected randomly.

(c) *SQL slammer worm attack*.
The IP protocol type is UDP, the destination port is set to 1434 and the packet size is between 371 and 400 bytes. Other attributes are selected randomly.

(d) *Nominal attack*.
All attributes in the packets are selected randomly in smaller value ranges, which contain the most frequently-appeared values of this attribute in the non-attack period. This attack supposes that the attackers know the value distributions of the single attributes and mimic it to carry out an attack.

(e) *Mixed attack*.
In this attack, the attack type of each packet will be selected randomly from the four types above.

The score calculating and packet discarding of CBF are not affected by the intensity of the attack and the changing frequency of the attack types. Thus in this evaluation, we will not largely focus the tests of CBF on changing the type and intensity of attacks like [7,8].

6.2. Simulation result and analysis

Fig. 8(a) shows the score distribution of generic attack and the legitimate flow using more than 100,000 packets of data. To avoid the trouble with decimal scores, we multiply the original CBF scores with 10,000 when shown in the graph. Since the legitimate attribute pair characteristics cannot be easily copied, most generic attack packets only have scores which consist of basic confidence *minconf*, 0.0005. For legitimate packets, high scores around 20–100 take place because they have more frequently-appeared attribute value pairs. Fig. 8(b) shows the cumulative distribution function (CDF) of the CBF scores of nominal attack and the legitimate flow (generic attack is not chosen here because its CDF curve is too steep to see a clear distribution). It illustrates more clearly that the majority of attack packets are concentrated in the low-score region.

Table 9
The performance of CBF and PacketScore under different attack types.

Attack type	Attack intensity	False positive rate (%)		False negative rate (%)	
		CBF	PacketScore	CBF	PacketScore
Generic	5×	0.513	3.266	0.695	0.0173
	10×	0.516	1.729	0.692	0.0432
TCP-SYN flood	5×	7.701	3.571	7.775	1.249
	10×	7.703	1.956	7.770	1.542
SQL slammer worm	5×	1.521	3.473	3.883	0.000
	10×	1.524	1.988	3.881	0.000
Nominal	5×	5.229	5.032	6.925	9.519
	10×	5.234	2.929	6.915	13.462
Mixed	5×	4.564	4.771	6.524	7.601
	10×	4.565	2.653	6.524	9.543

To evaluate CBF in a more precise way, we will test its performances of false positive (FP) rate and false negative (FN) rate when filtering. Both CBF and the classic scheme PacketScore are filtering methods that use attributes in TCP and IP headers to build *nominal profile*. Score packets are used to distinguish attack ones from legitimate ones. Here, we make some comparisons when highlighting the ability of CBF.

Table 9 shows the result of their performances. The *discarding threshold* values for discarding in CBF are chosen to give the best performance among all possible ones. Since the CBF scores are not affected by attack intensity, the FP and FN rates are almost the same when there are 5 times and 10 times amount of attack packets than normal.

In most cases, these two methods share similar filtering abilities. In generic attack, CBF has a lower false positive rate because it is quite hard to generate the accurate attribute value pairs in a random approach. In false negative rate, PacketScore has a better performance in SQL slammer worm attack but a worse one in mixed attack.

In TCP-SYN flood, the performance of CBF has some degradation. It results from the situation that the TCP-SYN packets may be also frequent in legitimate time. But the approximate 7.7% false

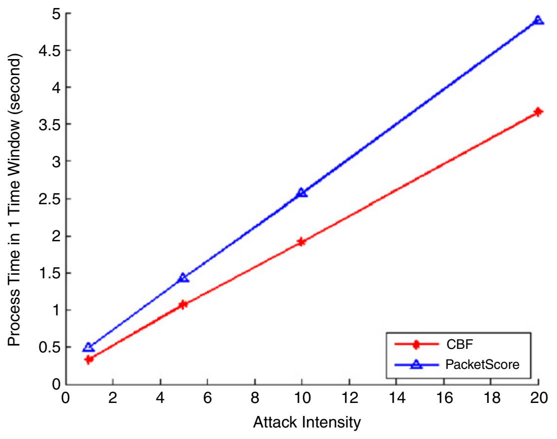


Fig. 9. Comparison of CBF and PacketScore methods in process time in the attack period.

positive rate and false negative rate can also be considered as an effective filtering in practice.

PacketScore has a worse performance in false negative rate in nominal attack compared to our method. This is because we assume the attackers have the information of the single attribute value distributions in this attack. For CBF, its filtering can be ineffective if the attackers find the correlation characteristics of the attribute pairs, but these data are quite impossible to be fully collected in practice.

At attack period, CBF are quite faster than PacketScore due to the simplicity of score calculation. Fig. 9 shows the process time in one time window (5 s) in the attack period for CBF and PacketScore. The attack intensity in the figure refers to the ratio of the amount of attack packets to the amount of non-attack packets in our experiment. Since CBF has no concept of time window in the attack period, we measure the time that CBF processes the same amount of packets as those in a 5 s window of PacketScore instead. Due to the limitation of our experiment environment, we believe that the process time in the figure for both methods can be reduced largely by optimizations and hardware supports.

Since the discarding period of PacketScore requires packet counting and CDF calculating, its process time in the figure under all attack intensity conditions is higher than that of CBF, which only needs a few looking-ups to generate a CBF score. For CBF, the most costly operation is to search the confidence values in *nominal profile*, so it can still be faster if a better hash function is adopted.

7. Case study: CBF in the 2014 Youth Olympic Games

In 2014, the second Youth Olympic Games will be hold in Nanjing, China. The organization issues are planed to be promoted with a Game Cloud platform and a Commit Cloud platform. In order to ensure the security of network services in the event, our method, Confidence-Based Filtering, is selected as one of the candidate DoS/DDoS attack defense plans in cloud deployment.

For usage, we develop an application based on CBF, named Cmd-CBF, for Linux OS environment. Cmd-CBF is written in C++, and the users can type in command lines to control it. Fig. 10 depicts the module design of Cmd-CBF. In this section, we will introduce the modules one by one, especially focusing on the implementation details not covered before, to illustrate how to use CBF in practice.

7.1. Packet capture and judgment

In both the attack period and non-attack period, CBF needs to capture incoming packets and then judge the legitimacy of them. In Cmd-CBF, we use libnetfilter_queue package provided by netfilter [23] in the Linux OS kernel (Linux 2.4.*x* and 2.6.*x* series). This package provides an API to process the packets

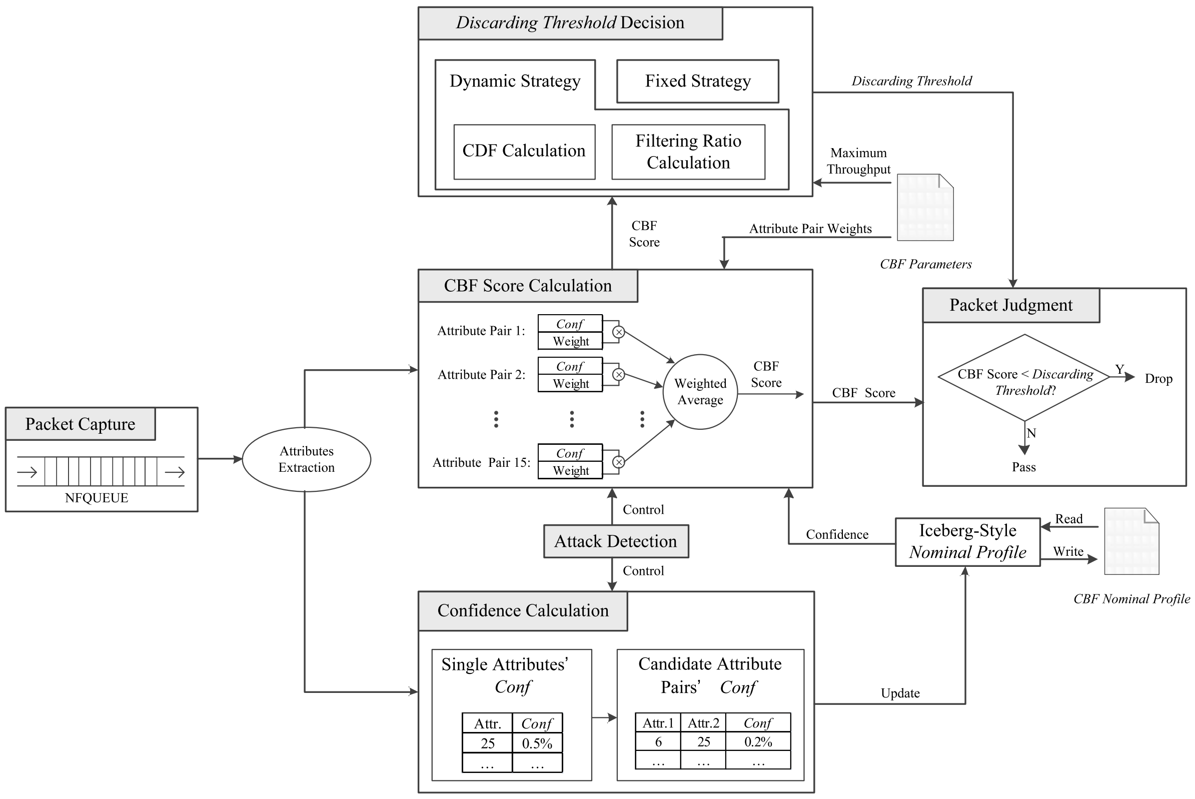


Fig. 10. Module design of Cmd-CBF for 2014 Youth Olympic Games in Nanjing.

that have been queued in NFQUEUE by the kernel packet filter. Before running Cmd-CBF, iptables command should be used to instruct the incoming packets to flow into NFQUEUE, waiting for judgments. Then, once a packet is received, the attribute extraction function of CBF will be triggered, and the rest of the CBF method follows.

When in need of packet legitimacy judgment, Cmd-CBF uses the verdict mechanism in libnetfilter_queue. In the non-attack period (Section 4), since the packets are used to generate *nominal profile*, we simply allow all of them to pass by marking them with NF_ACCEPT. In the attack period (Section 5), using the discarding threshold as a borderline, we mark the legitimate packets with NF_ACCEPT and the attack ones with NF_DROP. Instructed by these marks, the packets in NFQUEUE are judged successfully.

7.2. Attack detection

Attack Detection is a very important module which acts like a switch between the attack period and non-attack period of the CBF method. As designed, this function should be implemented by a DDoS detection method such as [4–6] in the front-end of the defense system. Nevertheless, since the detection method in the project has not been clearly decided, in Cmd-CBF we provide a naïve but feasible detection method for exclusive usage of it.

In this detection method, we use the max throughput of the network system as the detection criterion, since high volume of packets is the most important feature of DDoS attacks. We also set an inspection window to smooth the switch process. If the average throughput in the inspection window is greater than the max throughput, Cmd-CBF switches to the attack period of the CBF method. On the other hand, if the average throughput is less, we switch to the non-attack period. Actually, in Cmd-CBF we use two inspection windows in the detection method, one for attack-to-non-attack switch, and the other one for non-attack-to-attack switch. In this way, Cmd-CBF can have different response speeds for different switch directions, which increases the flexibility in usage.

7.3. CBF core modules

CBF core modules refer to the three blocks in the middle column in Fig. 10. They are CBF score calculation (Section 5.1), *discarding threshold* decision (Section 5.2) and confidence calculation (Section 4). In Cmd-CBF, the CBF score calculation part has nothing new compared to the former sections, which involves a weighted average operation of the attribute pairs' confidence values.

In *discarding threshold* decision module, Cmd-CBF implements both the dynamic and fixed strategies. Fixed strategy is much faster but the threshold can only be set properly by users who are familiar with the network environment and CBF method. So for more foolproof usage, we also develop the dynamic strategy module, which takes the max throughput as input and makes a *discarding threshold* decision via CDF calculation (see Section 5.2 for more details).

In dynamic strategy, we use the *discarding threshold* generated by the last window to judge the packets in the current window. In this way, once a packet arrives, Cmd-CBF can accept or drop it directly, instead of waiting until the threshold calculation for this window finishes. However, this incurs the problem about the first window: at the moment when Cmd-CBF switches from non-attack period to attack period, the incoming packets belong to the first window in the attack period, and no *discarding threshold* is prepared for their judgment. To solve it, we extract some packets from the previous window (in non-attack period), calculate their CBF scores and use CDF calculation to obtain the *discarding threshold* in need. This process slows down the judgment of the first packet in the first window after the non-attack-to-attack switch, and we name its time cost Attack Adapting Time. This time cost can affect the attack response performance of the CBF method, so in Cmd-CBF, we try to limit it by using a time counter when extracting packets from the last window. We include this time counter in the parameter setting of Cmd-CBF, thus users can set them according to their needs.

In the confidence calculation module, Cmd-CBF implements the 4 steps in Fig. 5 serially, instead of the pipeline process depicted in Fig. 6. For hash function used in the ice-berg style *nominal profile*, we choose the hash_map class provided by Standard C++ Library.

7.4. Parameter and nominal profile

As depicted in Fig. 10, Cmd-CBF uses two main external files: *CBF Parameters* and *CBF Nominal Profile*. *CBF Parameters* file is used for setting the parameters for Cmd-CBF. When the application starts, it will first read the parameters from this file using a predefined format. When the application stops or the user directly instructs, it will rewrite this file to record the current parameter settings. In Cmd-CBF, we limit that the reading and writing of the file *CBF Parameters* can only be used when the CBF method is not running, which prevents the possible faults caused by changing some critical parameters. For each parameter, Cmd-CBF also provides some fault-tolerant mechanisms for invalid input values.

In Cmd-CBF, *CBF Parameters* file includes the following parameters for user editing: the attack-to-non-attack switch and the non-attack-to-attack switch inspection time (Section 7.2), *minconf*, max throughput, fixed *discarding threshold*, the Attack Adapting Time counter (Section 7.3), and the weights for the 15 attribute pairs.

On the other hand, *CBF Nominal Profile* file is designed for input and output of *nominal profile*. In Cmd-CBF, the file first includes the *minconf* of the current *nominal profile*, then the attribute pair values and their confidence values one by one. When reading the file, Cmd-CBF will first check whether the *minconf* in the file is compatible with the one in use. If *minconf* in the file is smaller, Cmd-CBF will pick out the confidence values in the file which are above the current *minconf* to update the *nominal profile*. If the *minconf* in the file is bigger, some confidence values in that file are missing for updating, so Cmd-CBF will ask the users to choose whether to insist on updating with the incomplete file or to use one of them and discard the other one.

7.5. Command design

Cmd-CBF uses command lines in the terminal to interact with users. The manual for usage is shown in Fig. 11.

The startcbf and stopcbf commands are used to control whether reads the incoming packets from the NFQUEUE. When startcbf is instructed, Cmd-CBF will trigger the attribute extraction module, and the whole CBF method will start to function. Once stopcbf is instructed, it merely pauses the running method, instead of exiting the command line. To leave Cmd-CBF, the quit command should be used.

The check command is designed for check some running information about Cmd-CBF. When checking the cbfparameter, Cmd-CBF lists all parameters used in the CBF method, including some constant values which are not in *CBF Parameters* for user editing. The cbfstatus reports whether the CBF method is in attack or non-attack period. For watching the profile generation or CBF score calculation in real time, users can start livelog, which can be quit by typing "q" in the keyboard.

The renew command is designed for updating or clearing some settings or files. In Cmd-CBF we offers in this command the updating of parameters from *CBF Parameters* file, the clearing of the previous running logs in *CBFLog* file (an external file for recording

```
1. startcbf
   Description: start capturing packets and CBF method.
2. check [ cbfparameter | cbfstatus | livelog ]
  4.1. cbfparameter
       Description: check the current CBF parameter setting.
  4.2. cbfstatus
        Description: check the current CBF running status.
  4.3. livelog
        Description: check the CBF running details live. Press "q" to
       stop.
3. renew [ cbfparameter | log | nominalprofile ]
  4.1. cbfparameter
       Description: update the parameter settings from file.
  4.2. log
       Description: clear log.
  4.3. nominalprofile
       Description: clear nominal profile.
4. set [ cbfmode | autofiltering | dynamicthreshold] [ 0 | 1 ]
  4.1. cbfmode
       Description: set the CBF running mode.
       Value: 0 -- non-attack mode; 1 -- attack mode
  4.2. autofiltering
       Descripiton: set Filtering Mode
       Value: 0 -- manual filtering; 1 -- automatically filtering
  4.3. dynamicthreshold
       Descripiton: set Discarding Threshold Decision Strategy
       Value: 0 – fixed; 1 -- dynamic
5. stopcbf
   Description: stop running CBF.
6. quit
   Description: quit Cmd-CBF.
7. help
   Description: show how to use CBF method.
```

Fig. 11. Command line design of Cmd-CBF.

the functioning of Cmd-CBF), and the clearing of the current *nominal profile*. Here we do not provide users with the updating of *nominal profile* from some self-selected *CBF Nominal Profile* files. This is because the *nominal profile* is unique for different end systems as explored in Section 2, and each user should either stick to their own *nominal profile* used before or generate a new one if no suitable one exists.

The set command is used to change some running status of the CBF method. In Cmd-CBF, we provide the setting of attack or non-attack period, automatically or manually filtering, and fixed or dynamic *discarding threshold* decision. The setting of the parameters is achieved with the help of external file CBF Parameters, so in the set command we leave out this function.

As a result, the help command could provide the users with a full instruction of the command line usage, external files reading and writing, and explanations of the terms and expressions used in Cmd-CBF.

8. Discussion and analysis

CBF utilizes the attribute value pairs in TCP and IP headers to construct correlation characteristics. In Section 6, these characteristics are tested to be effective in distinguishing attack packets from legitimate ones under different types of denial-of-service attack. As shown in evaluation results, the most outstanding advantages of CBF are its high efficiency in the attack period and small storage requirements for *nominal profile*. These features make CBF powerful especially in attacks with an extremely large amount of traffic. In filtering ability, CBF does not have a strictly high accuracy compared to the previous researches. But the FP and FN rates at present are no more than 8%, which has already been acceptable in most cases.

Indeed, CBF can be ineffective if the attack packet flows mimic the correlation characteristics of legitimate flows. However, in order to carry out large quantities of packets as fast as possible, even finding out the value distribution of single attributes will be too costly for the attacker. Thus the case where attackers have the complete attribute pair distributions will not be quite possible in practice. The situation that the single attribute value distribution is known by attackers is simulated in nominal attack in Section 6 and CBF takes on a good performance by maintaining FP and FN rates around 5%–6%.

In the situation where a distributed attack is carried out, all the source IP addresses will not be spoofed in the attack flow. But CBF can still successfully defeat this attack because the ability of CBF depends on the co-appearance of two attributes. That means even if the source IP address is authentic, the attack packets need to have the right attribute which frequently appears along with that source IP address as well. Considering the difficulty of that, CBF will also be quite effective when dealing with distributed attacks.

Flash crowds are the situations where a large number of legitimate customers happen to visit a server at the same time period. For CBF, it will not confuse flash crowds with denial-of-service attack. Since the filtering of CBF will not be affected by the number of packets, the packets sent by legitimate customers will have the frequent correlation characteristics as usual. Thus these packets will also get a high CBF score to avoid being blocked.

9. Conclusion and future work

9.1. Conclusion

The key concept of CBF is correlation characteristic, which is the co-appearance of attribute pairs in our implementation. We introduced confidence to represent the distribution of attribute value pairs and then devised a feasible approach to generate the *nominal profile* in order to store these confidence values. With the *nominal profile*, CBF can calculate scores for incoming packets in the attack period to conduct filtering. Since the confidence reflects the frequency of appearances of the attribute value pairs, packets with more attribute value pairs of higher confidence will get a higher score, which means more legitimacy in this method. As shown in Sections 3 and 6, CBF has a small storage size, an acceptable filtering accuracy, and a high scoring speed, which together make it a practical DDoS defending method in cloud platforms.

9.2. Future work

In the future, a more flexible discarding strategy to set the *discarding threshold* is required. The candidate one should not be so time-consuming that CBF loses its advantage of fast response in the attack period. Also we will work on a more theoretical way of choosing the weights for each attribute pairs in CBF score calculation. The ideal strategy is adjusting the weights automatically based on the condition of the network. Finally, some optimizations and a better hash algorithm should be adopted to further accelerate the speed and the filtering accuracy of CBF.

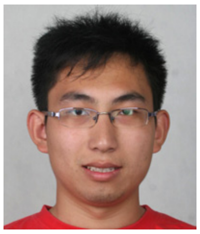
Acknowledgments

The authors would like to express their thanks for the plentiful and valuable help given by Liu Zhenxing. This paper is partly supported by the National Science Foundation of China under Grant No. 61021062, 61073032 and 60736015, and Jiangsu Provincial NSF Project under Grants BE2011171.

References

[1] M. Armbrust, et al., A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.
[2] L. Zhang, Q. Zhou, CCOA: cloud computing open architecture, in: Proceedings of the IEEE International Conference on Web Services, 2009, pp. 607–616.
[3] T. Peng, C. Leckie, K. Ramamohanarao, Survey of network-based defense mechanisms countering the DoS and DDoS problems, ACM Comput. Surv. 39 (1) (2007) 3.
[4] A. Chonka, J. Singh, W. Zhou, Chaos theory based detection against network mimicking DDoS attacks, IEEE Commun. Lett. 13 (9) (2009) 717–719.
[5] Y. Xiang, K. Li, W. Zhou, Low-rate DDoS attacks detection and traceback by using new information metrics, IEEE Trans. Inf. Forensics Secur. 6 (2) (2011) 426–437.
[6] H. Liu, M.S. Kim, Real-time detection of stealthy DDoS attacks using time-series decomposition, in: Communications (ICC), 2010 IEEE International Conference, 2010.
[7] Y. Kim, W.C. Lau, M.C. Chuah, H.J. Chao, Packetscore: a statistics-based packet filtering scheme against distributed denial-of-service attacks, IEEE Trans. Dependable Secure Comput. 3 (2) (2006) 141–155.
[8] P.E. Ayres, H. Sun, H.J. Chao, W.C. Lau, ALPi: a DDoS defense system for high-speed networks, IEEE J. Sel. Areas Commun. 24 (10) (2006) 1864–1876.
[9] H. Wang, C. Jin, K.G. Shin, Defense against spoofed IP traffic using hop-count filtering, IEEE/ACM Trans. Netw. 15 (1) (2007) 40–53.
[10] P. Du, A. Nakao, DDoS defense deployment with network egress and ingress filtering, in: Proceeding of the IEEE International Conference on Communications, 2010, pp. 1–6.
[11] Z. Duan, X. Yuan, J. Chandrashekar, Controlling IP spoofing through interdomain packet filters, IEEE Trans. Dependable Secure Comput. 5 (1) (2007) 22–36.
[12] F. Soldo, A. Markopoulou, K. Argyraki, Optimal filtering of source address prefixes: models and algorithms, in: Proc. IEEE INFOCOM, 2009.
[13] M.T. Goodrich, Probabilistic packet marking for large-scale IP traceback, IEEE/ACM Trans. Netw. 16 (1) (2008) 15–24.
[14] Y. Xiang, W. Zhou, M. Guo, Flexible deterministic packet marking: an IP traceback system to find the real source of attacks, IEEE Trans. Parallel Distrib. Syst. 20 (4) (2009) 567–580.
[15] S. Yu, W. Zhou, R. Doss, W. Jia, Traceback of DDoS attacks using entropy variations, IEEE Trans. Parallel Distrib. Syst. 22 (3) (2011) 412–425.
[16] Y. Xie, S. Yu, Monitoring the application-layer DDoS attacks for popular websites, IEEE/ACM Trans. Netw. 17 (1) (2009) 15–25.
[17] Y. Xie, S. Yu, A large-scale hidden semi-Markov model for anomaly detection on user browsing behaviors, IEEE/ACM Trans. Netw. 17 (1) (2009) 54–65.
[18] W. Dou, L. Qi, X. Zhang, J. Chen, An evaluation method of outsourcing services for developing an elastic cloud platform, J. Supercomput. (2010) http://dx.doi.org/10.1007/s11227-010-0491-2. published online.
[19] MAWI Traffic Archive [Online]. Available: http://tracer.csl.sony.co.jp/mawi/.
[20] H. Jiawei, M. Kamber, Data Mining: Concepts and Techniques, second ed., Morgan Kaufmann Publishers, 2011.
[21] B. Babcock, et al. Models and issues in datastream systems, in: ACM Symp. Principles of Database Systems, 2002.
[22] S. Kasera, et al. Fast and robust signaling overload control, in: Proc. Int'l Conf. Network Protocols, 2001.
[23] netfilter/iptables project [Online]. Available: http://www.netfilter.org/.

Wanchun Dou received his Ph.D. degree in Mechanical and Electronic Engineering from Nanjing University of Science and Technology, China, in 2001. Now, he is a Full Professor of the State Key Laboratory for Novel Software Technology, Nanjing University, China. From Apr. 2005 to Jun. 2005 and from Nov. 2008 to Feb. 2009, he respectively visited the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, as a visiting scholar. Up to now, he has chaired three NSFC projects and published more than 60 research papers in international journals and international conferences. His research interests include workflow, cloud computing and service computing.



Qi Chen got his B.S. degree from the Computer Science and Technology department in Nanjing University in 2012. During 2011–2012, he also worked on network security as a guest research assistant, co-supervised by Professor Wanchun Dou from Nanjing University and Lecturer Shui Yu from Deakin University, Australia. He is now a Ph.D. student supervised by Professor Z. Morley Mao in the EECS department of the University of Michigan at Ann Arbor.

Jinjun Chen is an Associate Professor from the Faculty of Engineering and IT, University of Technology Sydney, Australia. He holds a Ph.D. in Computer Science and Software Engineering (2007) from Swinburne, a master degree in Engineering (1999) and a bachelor degree in Applied Mathematics (1996) from Xidian University, China. Dr. Chen's research interests include cloud computing, social computing, green computing, service computing, e-science/e-research and workflow management. His research results have been published in more than 100 papers in high quality journals and at conferences, including the ACM Transactions on Software Engineering and Methodology (TOSEM), IEEE Transactions on Software Engineering (TSE), and the International Conference on Software Engineering (ICSE). He received the Swinburne Vice-Chancellor's Research Award for early career researchers (2008), IEEE Computer Society Outstanding Leadership Award (2008–2009), IEEE Computer Society Service Award (2007) and the Swinburne Faculty of ICT Research Thesis Excellence Award (2007).