Chapter 2
Size Constraint Group Testing and DoS Attacks

Abstract In this chapter, we introduce the first application of group testing in detecting application Denial-of-Service (DoS) attack , which aims at disrupting application service rather than depleting the network resource. This attack has emerged as one of the greatest threat to network services. Owing to its high similarity to legitimate traffic and much lower launching overhead than classic DoS attack, this new assault type cannot be efficiently detected or prevented by existing detection solutions. To identify application DoS attack, we present a novel group testing (GT)-based approach deployed on back-end servers, which not only offers a theoretical method to obtain short detection delay and low false positive/negative rate, but also provides an underlying framework against general network attacks. This new application requires a new class of group testing, called size constraint group testing.

2.1 Overview

One of the most critical problems in Internet security is the Denial-of-Service (DoS) attack , which aims to make a service unavailable to legitimate clients [1]. DoS attacks mainly abuse the network bandwidth around the Internet subsystems and degrade the quality of service by generating congestions at the network [1, 2]. However, with the boost in network bandwidth and application service types recently, the target of DoS attacks have shifted from network to server resources and application procedures themselves, forming a new application DoS attack [3].

By exploiting the flaws in application design and implementation, application DoS attacks exhibit three advantages over traditional DoS attacks which help evade normal detections: malicious traffic is always indistinguishable from normal traffic, adopting automated script to avoid the need for a large amount of "zombie" machines or bandwidth to launch the attack, much harder to be traced due to multiple redirections at proxies. According to these characteristics, the malicious traffic can be classified into legitimate-like requests of two cases: (1) at a high interarrival rate, (2) consuming more service resources. We call these two cases “high-rate” and “high-workload” attacks, respectively.

Since malicious requests can be made arbitrarily similar to legitimate ones, accurately and efficiently distinguishing them is a challenging task for any defense mechanism. In addition, these attacks usually do not cause congestion at the network level, thus bypassing the network-based defense systems [1]. As a result, efficiently identifying the attackers in application DoS attacks, which is the focus of this chapter, becomes much more difficult.

Consequently, several network-based defense methods have tried to detect these attacks by controlling traffic volume or differentiating traffic patterns at the intermediate routers [4–9]. However, these approaches aim to defend at the network layer, i.e, differences in traffic patterns, traffic volume, which the application DoS attacks can bypass. Moreover, each traffic is inspected against the normal behaviors model, thereby increasing the time complexity. Because of that, detection and mitigation at the end-system of the victim servers have been proposed [3, 10, 11]. Among them the DDoS shield [3] and CAPTCHA-based defense [10] are the representatives of the two major techniques of system-based approaches: session validation based on legitimate behavior profile and authentication using human-solvable puzzles. In [3], each session is inspected against a normal behaviors model (i.e. interarrival time distribution) to detect a suspected session. Based on this suspicion level, a set of abnormal requests is enumerated. Note that the method keeps state for each session, which grows linearly with the number of clients. In [10], the service requires clients to solve puzzles and eliminates those who send a wrong solution at a high rate. However, this method is restricted to services with human clients who can solve puzzles. Plus, it introduces additional service delays for legitimate clients. It also checks each answer from clients one by one.

As the victim’s resources will be exhausted, detecting application DoS attacks is always possible based on the performance of the attacked resources. Thus the key problem to identifying attackers lies in how to group clients together on each server so that if a server is under attack, we can quickly identify these attackers without examining each request. This problem resembles the group testing theory, therefore, the GT technique is uniquely suitable for the detection of application DoS attackers with high accuracy and efficiency. By utilizing the GT technique, attackers are detected based merely on the performance of attacked resources, without the need to keep track of attack signatures, tightly specify legitimate behavior, or examine each request one by one. Thus the GT-based defense scheme may overcome the limitations of current detection approaches.

The realization of this approach, however, raises several challenges. The classical GT theory assumes that each pool can have as many items as needed and the number of pools for testing is unrestricted. However, in order to provide real application services, servers cannot have infinite quantity or capacity, that is, the number of available pools is given and we cannot freely assign many clients/requests into a sever (pool). Thus constraints on these two parameters are required to complete the testing model, which we call Size Constraint Group Testing.

In a system viewpoint, the defense scheme is to embed multiple virtual servers within each physical back-end server, and map these virtual servers to the testing pools in GT, then assign clients into these pools by distributing their service requests to different virtual servers. By periodically monitoring some indicators (e.g., average responding time) for resource usage in each server, and comparing them with some dynamic thresholds, all the virtual servers can be judged as "safe" or "under attack". By means of the decoding algorithm of GT, all the attackers can be identified. Therefore, the biggest challenges of this method are threefold: (1) How to construct a testing matrix to enable prompt and accurate detection. (2) How to regulate the service requests to match the matrix, in a practical system. (3) How to establish proper thresholds for server source usage indicator to generate accurate test outcomes.

In the rest of this chapter, we first present the attacker and detection models to provide more information above the network security setting. The above three questions are addressed in Sects. 2.4 and 2.5.

2.2 Network System Models

2.2.1 DoS Attacker Model

The maximum destruction caused by the attacks includes the depletion of the application service resource at the server side, the unavailability of service access to legitimate user, and possible fatal system errors which require rebooting the server for recovery. Any malicious behaviors can be discovered by monitoring the service resource usage, based on dynamic value thresholds over the monitored objects. We also assume that application interface presented by the servers can be readily discovered and clients communicate with the servers using HTTP/1.1 sessions on TCP connections [3]. We consider a case that each client provides a non-spoofed **ID** (e.g. SYN-cookie [12]), which is utilized to identify the client during our detection period. Owing to the characteristics of this DoS attack, we can assume that the number of attackers $d \ll n$, where n is the total client amount.

2.2.2 Victim/Detection Model

The victim model in the GT-based defense scheme consists of multiple back-end servers, which can be web/application servers, database servers, and distributed file systems. We do not take classic multi-tier web servers as the model, since the detection scheme is deployed directly on the victim tier and identifies the attacks targeting at the same victim tier, thus multi-tier attacks should be separated into several classes to utilize this detection scheme. The victim model along with front-end proxies are shown in Fig. 2.1.

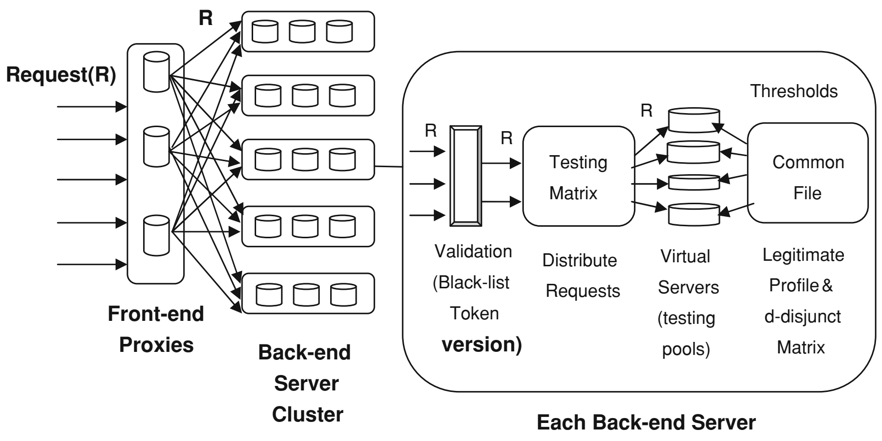

Fig. 2.1 The victim and detection models. This figure also provides an overview of the GT-based detection scheme.

We assume that all the back-end servers provide multiple types of application services to clients using HTTP/1.1 protocol on TCP connections. Each back-end server is assumed to have the same amount of resource. Moreover, the application services to clients are provided by K virtual private servers (K is an input parameter), which are embedded in the physical back-end server machine and operating in parallel. Each virtual server is assigned with equal amounts of static service resources, e.g., CPU, storage, memory, and network bandwidth. The operation of any virtual server will not affect the other virtual servers in the same physical machine. The reasons for utilizing virtual servers are twofold: first, each virtual server can reboot independently, thus is feasible for recovery from possible fatal destruction; second, the state transfer overhead for moving clients among different virtual servers is much smaller than the transfer among physical server machines.

As soon as the client requests arrive at the front-end proxy, they will be distributed to multiple back-end servers for load balancing. On being accepted by one physical server, one request will be simply validated based on the list of all identified attacker **ID**s (black-list). If it passes the authentication, it will be distributed to one virtual server within this machine by means of virtual switch. This distribution depends on the testing matrix generated by the detection algorithms described in Sect. 2.4. By periodically monitoring the average response time to service requests and comparing it with specific thresholds fetched from a legitimate profile, each virtual server is associated with a “negative” or “positive” outcome and a decision over the identities of all clients can be made among all physical servers, as discussed further in Sect. 2.5.

As can be seen, the major components of this model are (1) the Testing Matrix based on which we decide the distribution of clients/requests to virtual servers so that a test can be performed and (2) how to determine the testing results.

2.3 Size Constraint Group Testing

As mentioned in the detection model, each testing pool is mapped to a virtual server within a back-end server machine. Although the maximum number of virtual servers can be extremely huge, since each virtual server requires enough service resources to manage client requests, it is practical to have the virtual server quantity (maximum number of servers) and capacity (maximum number of clients that can be handled in parallel) constrained by two input parameters K and w respectively. Thus the testing matrix must satisfy the two mentioned constraints, thereby extending the traditional GT model as follows:

Definition 2.1 *Size Constraint Group Testing* (*SCGT*): For any binary matrix M, let $w_{\mathrm{i}} = \sum_{j=1}^{n} M[i, j]$ be the weight of the *ith* pool, and t be the number of pools. The model is for identifying d defected items in the minimum period of time, by constructing matrix $M_{\mathrm{t}\times\mathrm{n}}$ and conducting group testing procedures based on that, where $w_i \leq w$ for a given w and $t \leq K$ for a given server quantity K.

The following section presents the solutions to construct such a testing matrix.

2.4 Matrix Construction and Latency Analyses

We present three detection algorithms: sequential detection with packing (SDP), sequential detection without packing (SDoP) and partial non-adaptive detection (PND) in this section. Note that the length of each testing round is a predefined constant P (which is discussed later), hence we analyze the algorithm complexity in terms of the number of testing rounds for simplicity.

Since each client has a unique **ID**, the two terms "**ID**" and "client" are interchangeable in this section. In addition, all the following algorithms are executed in each physical back-end server, which is an independent testing domain as mentioned. Therefore, the term "servers" denote the K virtual servers in this section.

For simplicity, we assume that $n \equiv 0\ (\mathbf{mod}\ K)$, $n > w \geq d \geq 1$ and $|A| \geq d+1$. Notice that the last inequality holds in practice as the properties of application DoS attacks indicate in Sect. 2.1.

2.4.1 Sequential Detection With Packing

This algorithm investigates the benefit of classic sequential group testing, i.e., optimizing the grouping of the subsequent tests by analyzing existing outcomes. Similar to traditional sequential testing, each client (column) only appears in one testing pool (server) at a time. However, to make full use of the available K servers, we have all servers conduct tests in parallel. Details can be found in Algorithm 4.

Algorithm 4 Sequential Detection with Packing (*SDP*)

```
1: while $|S| \neq 0$ do
2:     for all server $i$ in $A$ do
3:         $w_{\mathrm{i}} \leftarrow \lceil \frac{|S|}{|A|} \rceil$ // Assign even number of distinct clients to each server.
4:     end for
5:     $G \leftarrow \lceil \frac{n-(|S|-(|A \setminus I|)w_{\mathrm{i}})}{w} \rceil$ // Identified legitimate IDs exempt from the
       following tests. Their subsequent requests are handled by $G$ non-testing
       servers, which are selected from the $K$ servers and only provide normal
       services (no testing).We call this process as "packing" the clients into
       non-testing servers.
6:     for all server $i$ under attack do
7:         $Q \leftarrow$ set of IDs on $i$
8:         if $|Q| = 1$ then
9:             $S \leftarrow S \setminus Q$ // This ID belongs to an attacker. Add it into the black-list.
10:        end if
11: end for
12: $S \leftarrow S \setminus L$ // All IDs on safe (with negative test outcome) servers are identified
    as legitimate.
13: $A \leftarrow$ {server1, ..., server $(K-G)$} // Select the first $G$ servers as non-testing
    machines.
14: end while
```

2.4.1.1 Algorithm Description

The basic idea of the *SDP* algorithm can be sketched as follows. Given a set S of suspect **ID**s, the algorithm first randomly assigns (i.e. distribute their requests) them to the available testing servers in set A, where each server will receive requests from approximately the same number of clients, roughly $w_{\mathrm{i}} = \lceil \frac{|S|}{|A|} \rceil$. For each test round, the **ID**s on the negative servers are identified as legitimate clients, and are "packed" into a number G of non-testing machines. Since they need no more tests, only normal services will be provided for the following rounds. As more testing servers will speed up the tests, given at most K server machines in total, G is then supposed to be minimized to the least, as long as all identified legitimate clients can be handled by the non-testing w-capacity servers. Hence $G = \lceil \frac{n-(|S|-(|A \setminus I|)w_i)}{w} \rceil$.

With the assumption $|A| \geq d+1$, we have at least $|A|-d$ servers with negative outcomes in each testing round, hence at least $(|A|-d)w_{\mathrm{i}}$ legitimate **ID**s are identified. If any server containing only one active **ID** is found under attack, the only **ID** is definitely an attacker. Then its **ID** is added into the black-list and all its requests are dropped. Iterate the algorithm until all **ID**s are identified as either malicious or legitimate. Via the "packing" strategy, legitimate clients can exempt from the influence of potential attacks as soon as they are identified.

Fig. 2.2 An example of how SDP algorithm works

$$\begin{array}{c|cccccccccc}
 & \textcircled{1} & 2 & \textcircled{3} & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array} \xrightarrow{testing} \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{array}{c|cccccccccc}
 & \textcircled{1} & 2 & \textcircled{3} & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1
\end{array} \xrightarrow{testing} \begin{pmatrix} 1 \\ 0 \\ 1 \\ * \\ * \end{pmatrix}$$

$$\begin{array}{c|cccccccccc}
 & \textcircled{1} & 2 & \textcircled{3} & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\hline
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1
\end{array} \xrightarrow{testing} \begin{pmatrix} 1 \\ 0 \\ 1 \\ * \\ * \end{pmatrix}$$

Example A 3-round detection example in Fig. 2.2 illustrates the execution of this algorithm. Given $n = 10$, $d = 2$, $K = 5$, and $w = 4$. Let $|S| = 10$, $|A| = 5$, and **ID** 1,3 be the attackers. In the first round, $w_i = 2$ yields two **ID**s on each server. Server 1 and server 2 (containing **IDs** 1 and 3 respectively) will indicate being under attack, i.e., $|I| = 2$ (two servers are under attack) and $|A \setminus I| = 3$, pack legitimate **ID**s on the other three servers into $G = 2$ servers. Update $|S| = 4$ and $|A| = 3$. In the second round, $w_i = 2$ yields at most two **ID**s on each server, assume that server 1 and server 3 are under attack, then again pack the legitimate **ID** 4 in server 2 to server 4 (server 5 is already full). In the third round, servers 1 and 3 only contain attacker **IDs** 1 and 3, respectively. Therefore, the two attackers are identified at the end of this round. What we would like to point out is, in this instance, for every new assignment, no testing server contains multiple malicious **ID**, which will affect more servers and yield to the least number of safe servers at each round, and thus is likely to cost the most testing rounds. Hence, this is an instance of worst performance, and its result verifies our $O(\log_{\frac{K'}{d}} \frac{n}{d})$ upper bound mentioned in the below analysis.

2.4.1.2 Performance Analysis

With regard to the computational overhead of the matrix M in this context, it is a re-mapping of the suspect clients to the testing servers based on simple strategies and previous testing outcomes, thus is negligible and up to $O(1)$. The time cost of each testing round is at most P, including the recomputation time of the testing matrix.

Besides this, the overall time complexity of this algorithm in terms of the number of testing rounds are depicted in the following theorems.

Lemma 2.1 The number of available testing server machines is $|A| \in \left[K - \lceil \frac{n-d}{w} \rceil, K\right]$.

proof In each round, the total number of legitimate **ID**s $n - (|S| - (|A \setminus I|)w_{\mathrm{i}})$ is non-decreasing. Hence, the number of servers used for serving identified legitimate **ID**s, $G = \lceil \frac{n-(|S|-(|A \setminus I|)w_{\mathrm{i}})}{w} \rceil$ is non-decreasing. Therefore $|A| = K - G$ will finally converge to $K - \lceil \frac{n-d}{w} \rceil$. □

Theorem 2.1 *The SDP algorithm can identify all the d attackers within at most* $O(\log_{\frac{K'}{d}} \frac{n}{d})$ *testing rounds, where* $K' = K - \lceil \frac{n-d}{w} \rceil$.

proof In each round, at most d servers are under attack, i.e., at most $\frac{d|S|}{|A|}$ **ID**s remain suspect. Since at the end of the program, all legitimate **ID**s are identified, at most d malicious **ID**s remain suspect. If they happen to be handled by d different servers, i.e., each positive server contains exactly one attacker, then the detection is completed. If not, one more testing round is needed. Assume that we need at most j testing rounds in total, then $n\left(\frac{d}{|A|}\right)^{\mathrm{j}-1} = d$ where $K - \lceil \frac{n-d}{w} \rceil \leq |A| \leq K$. Therefore, $j \leq \log_{(\frac{K'}{d})} \frac{n}{d} + 1$, where $K' = K - \lceil \frac{n-d}{w} \rceil$. □

Remark Note that the packing assignment in the SDP algorithm reflects the following design principle: serve all legitimate clients with the least number of good servers (non-testing machines) in order to achieve the maximum number of testing servers, thereby minimizing the testing rounds. In addition, it also ensures that all legitimate requests can be directed to a safe server as soon as possible.

2.4.2 Sequential Detection Without Packing

Considering the potential overload problem arises from the "packing" scheme adopted in *SDP*, we present the SDoP algorithm of which legitimate clients do not shift to other servers after they are identified. This emerges from the observation that legitimate clients cannot affect the test outcomes since they are negative. Algorithm 5 includes the abstract pseudocode of this *SDoP* scheme.

Algorithm 5 Sequential Detection without Packing (*SDoP*)

```
1: $w_{\mathrm{i}} \leftarrow$ number of IDs on server $i$;
2: for all server $i$ do
3:     $w_{\mathrm{i}} \leftarrow \lceil \frac{S}{A} \rceil$ // Evenly assign clients to servers as SDP did.
4: end for
5:
6: while $|S| \neq 0$ do
```

```
 7:     Randomly reassign $|S|$ suspect **ID**s to $K$ servers, and keep legitimate
        **ID**s unmoved.
 8:     $L \leftarrow$ set of **ID**s on safe servers
 9:     $S \leftarrow S \setminus (S \cap L)$ // $|S \cap L|$ **ID**s are identified as legitimate.
10:     for all server $i$ under attack do
11:         $Q \leftarrow$ set of **ID**s on $i$
12:         if $|Q \cap S| = 1$ then
13:             $S \leftarrow S \setminus Q$// The clients in $Q \cap S$ are attacker and added into the
                black-list.
14:         end if
15:     end for
16:     for all servers $i$ do
17:         if $w_i = w$// The volume approaches the capacity. then
18:             Reassign all $n - |S|$ legitimate **ID**s to $K$ servers, and go to 6 // Load
                balancing.
19:         end if
20:     end for
21: end while
```

2.4.2.1 Algorithm Description

The basic idea of the *SDoP* algorithm can be sketched below. Given suspect **ID**s set S with an initial size n, evenly assign them to the K server machines, similar to *SDP* in the first round. For the following rounds, assign suspect **ID**s to the K servers instead of $|A|$ available ones. For the identified legitimate IDs, never move them until their servers are to be overloaded. In this case, reassign all legitimate **ID**s over the K machines to balance the load. For a server with the positive outcome, the **ID**s active on this server but not included by the set of identified legitimate ones are still be identified as suspect. However, if there is only one suspect **ID** of this kind in a positive server, this **ID** is certainly an attacker.

Remarks Compared with SDP, if no overloading happens, the SDoP algorithm avoids moving identified legitimate clients, thereby decreasing the reassignment expense and increasing the number of available testing machines. However, if w is relatively small, the algorithm also has to reassign all legitimate clients to K servers and balance the weight w_i on different servers. Once overloading happens, the reason for reassigning both legitimate and suspect ones into K servers at two different testing rounds, instead of reassigning them together at one time, is to avoid the possibility that all non-testing (safe) servers after the reassignment contain no suspect client and much more legitimate clients than others, i.e, a large w_i. We will analyze the performance of this algorithm in the following section with regard to different w scope.

Fig. 2.3 An example of how SDoP algorithm works

$$\begin{array}{cc}
 & \begin{array}{cccccccccc}\textcircled{1} & 2 & \textcircled{3} & 4 & 5 & 6 & 7 & 8 & 9 & 10\end{array}\\
\begin{array}{c}1\\2\\3\\4\\5\end{array} & \begin{pmatrix}1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0\\0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1\end{pmatrix}
\end{array}
\xrightarrow{\textit{testing}}
\begin{pmatrix}1\\1\\0\\0\\0\end{pmatrix}$$

$$\begin{array}{cc}
 & \begin{array}{cccccccccc}\textcircled{1} & 2 & \textcircled{3} & 4 & 5 & 6 & 7 & 8 & 9 & 10\end{array}\\
\begin{array}{c}1\\2\\3\\4\\5\end{array} & \begin{pmatrix}0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0\\1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0\\0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1\end{pmatrix}
\end{array}
\xrightarrow{\textit{testing}}
\begin{pmatrix}1\\0\\0\\1\\0\end{pmatrix}$$

Example Figure 2.3 contains a 2-round detection example to illustrate the execution of this algorithm. Assume $n=10$, $d=2$, $K=5$ and the client 1, 3 are the attackers. In the first testing round, $w_{\mathrm{i}} = 2$ yields two **ID**s on each server, then only servers 1 and 2 will have positive outcomes. Update suspect **ID** set $S = \{1, 2, 3, 4\}$. In the second testing round, since no attackers have been identified and no servers are overloaded, we reassign all suspect **ID**s to K servers while not moving the legitimate clients, and servers 1 and 4 turn out to be under attack. Hence the legitimate **ID** set is $L = \{2, 4, 5, 6, 7, 8, 9, 10\}$ and the two attackers are captured within two testing rounds.

2.4.2.2 Performance Analysis

It is trivial to see that the computational overhead of the testing matrix M is similar to that of the SDP algorithm, therefore it can be ignored. The following theorems exhibit the overall detection delay of the algorithm, in terms of the number of testing rounds.

Lemma 2.2 *The number of* **ID***s* w_{i} *on server i does not exceed server capacity w till round j, where* $j = \log_{\frac{\mathrm{K}}{\mathrm{d}}} \frac{n}{n-w(K-d)}$.

proof At most d servers get positive outcomes in each round. If we assume round j is the last round that has no overloaded servers, then the maximum number of **ID**s on one server at round j is: $w^{\mathrm{j}}_{\max} = \sum_{i=1}^{j} (\frac{n}{K})(\frac{d}{K})^{\mathrm{i}-1}$. Since round j+1 will have at least one overloaded server, we have $w^{\mathrm{j}}_{\max} = w$, then $j = \log_{\frac{\mathrm{K}}{\mathrm{d}}} \frac{n}{n-w(K-d)}$. □

Lemma 2.3 *All malicious* **ID***s are identified within at most* $O(\log_{\frac{K}{d}} \frac{n}{d})$ *testing rounds if* $w \geq \frac{n-d}{K-d}$.

proof In each round with no overloaded servers, at most $\frac{d}{K}|S|$ legitimate **ID**s remain suspect. After $i = \log_{\frac{K}{d}} \frac{n}{K}$ rounds at most K suspect **ID**s are left, so we need only one more round to finish the testing. Hence, we need $i + 1 = \log_{\frac{K}{d}} \frac{n}{d}$ rounds. Since it requires that no servers are overloaded within these rounds, we have $i + 1 < j = \log_{\frac{K}{d}} \frac{n}{n-w(K-d)}$ which yields $w \geq \frac{n-d}{K-d}$. □

Lemma 2.4 *All malicious* **ID***s are identified within at most* $O(\log_{\frac{K}{d}} \frac{n}{d})$ *testing rounds if* $1 \leq w \leq \frac{n-d}{K-d}$.

proof If $w \leq \frac{n-d}{K-d}$, reassignments for legitimate **ID**s will be needed after round $j = \log_{\frac{K}{d}} \frac{n}{n-w(k-d)}$. Assume that the algorithm needs x more testing rounds in addition to $i + 1 = \log_{\frac{K}{d}} \frac{n}{d}$ to reassign and balance legitimate **ID**s on all servers, since the position changes of these legitimate **ID**s will neither influence test outcomes nor delay the identifications of suspect set S (they take place simultaneously), we hence have

$$\begin{aligned} x &\leq i + 1 - j \\ &\leq \log_{\frac{K}{d}} \frac{n}{d} - \log_{\frac{K}{d}} \frac{n}{n - w(k-d)} \\ &\leq \log_{\frac{K}{d}} \frac{n - w(K-d)}{d} \leq \log_{\frac{K}{d}} \frac{n}{d} \end{aligned}$$

Therefore, the total number of testing rounds is at most $O(\log_{\frac{K}{d}} \frac{n}{d})$. □

Theorem 2.2 *The* SDoP *algorithm can identify all attackers within at most* $O(\log_{\frac{K}{d}} \frac{n}{d})$ *testing rounds.*

Proof Directly from Lemmas 2.3 and 2.4. □

2.4.3 Partial Non-Adaptive Detection

Considering the fact that in the two sequential algorithms mentioned, we could not identify any attackers until we isolate each of them to a virtual server with a negative outcome, which may increase the detection latency. Therefore, we present a hybrid of sequential and non-adaptive method in this section. In this scenario, the requests from the same client will be received and responded by different servers in a round-robin manner. Different from *SDP* and *SDoP*, a d-disjunct matrix is used as the testing matrix in this scheme and attackers can be identified without the need of isolating them into servers.

As mentioned in Chap. 1, matrix M is called d-disjunct if no single column is contained in the boolean sum of any other d columns, and all positive items can be identified within one round. Therefore we adopt this method as a subroutine to generate M for each testing round. For the following analysis, we refer to this algorithm as *dDisjunct* (*n*,*d*) (Algorithm 2), which takes *n,d* as input, and outputs a *d*-disjunct matrix with *n* columns.

We now introduce the *PND* detection method using *d*-disjunct matrix with an assumption that the row weight of the constructed matrix does not exceed the server capacity w. Let *t*(*d,n*) denote the number of rows required in the *d*-disjunct matrix with *n* columns obtained from Algorithm 2. We need to consider different ranges of input *K* with respect to *t*(*d,n*) as follows:

Case 1 $K \geq t(d,n)$. In this case, all *n* suspect **ID**s can be assigned to *t*(*d,n*) servers according to a *d*-disjunct matrix, and thus we can identify all malicious IDs with one testing round.

Case 2 $K < t(d,n)$. With inadequate servers to test all *n* clients in parallel, we can iteratively test a part of them, say $n' < n$, where $t(d,n') \leq |A|$. The detailed scheme is depicted as follows and the pseudocode is shown in Algorithm 6.

2.4.3.1 Algorithm Description

In the PND algorithm, all *n* suspect **ID**s are evenly distributed to all *K* servers as the other algorithms did at the beginning. After the first round, all identified legitimate **ID**s are packed into $G = \lceil \frac{n-|S|}{w} \rceil$ servers. Therefore, $|A| = K - G$ servers are available for testing afterwards. If $|A| \geq t(d,|S|)$, we can construct a *d*-disjunct matrix $M_{|A|\times|S|}$ by calling function $dDisjunct(|S|, d)$ and identify all **ID**s in one more testing round. Otherwise, if $|A| < t(d,|S|)$, we can first find the maximum *n*' ($1 \leq n' \leq |S|$) such that $t(d,n') \leq |A|$; secondly, partition set *S* into two disjoint sets S_1 and S_2 with $|S_1| = n'$; thirdly pack all S_2 to *G* non-testing server machines and call subroutine $dDisjunct(|S_1|, d)$ with updated *d* to identify all **ID**s in S_1; and finally drop the attackers and pack all legitimate **ID**s into *G* machines and continue to test $S = S_2$. Iterate this process until all **ID**s are identified.

2.4.3.2 Performance Analysis

The time complexity of the function *dDisjunct*(*n,d*) (Algorithm 2) is *O*(*sqn*), i.e. *O*(*Kn*) with $sq \leq K$, which can be decreased to *O*(1) time cost in real-time. Since it is too complicated to use the upper bound in Theorem 1.5 for *t*(*d,n*), in this analysis, we instead use a simple bound $t(d,n) = n$ to obtain a rough estimate on the algorithm complexity in terms of the number of rounds. Moreover, we further investigate its performance based on an optimal *d*-disjunct, so as to demonstrate the potential of this scheme.

Algorithm 6 Partial Non-adaptive Detection with $K < t(d,n)$

```
1: Evenly assign $n$ IDs to $K$ servers with no two servers having the same client
   IDs. Pack all legitimate IDs on into $G$ servers.
2: $|A| \leftarrow K - G$ // Update the number of testing servers.
```

```
3: if $|A| \geq t(d, |S|)$ then
4:     Run $dDisjunct(d, |S|)$ (Algorithm 2)
5:     decode from the obtained $d$-disjunct matrix and identify all IDs;
6:     $S \leftarrow 0$; // Testing finishes.
7: else
8:     while $|S| \neq 0$ do
9:         Partition $S$ into $S_1, S_2$ where $|S_1| = n'$ and $n' = \max n'' \in [1, |S|]$
           satisfying $t(d,n'') \leq |A|$.
10:        Run $dDisjunct(d, |S_1|)$, decode from the obtained $d$-disjunct matrix and
           identify all IDs in $S_1$, update $d$.
11:        Pack legitimate IDs in $S_1$ to $G$ machines, update $G$ and $A$; $S \leftarrow S_2$.//
           Iteration with suspect ID set $S_2$.
12:    end while
13: end if
```

Lemma 2.5 *The* PND *algorithm identifies all the attackers within at most* $O(dw^2/(Kw - w - n))$ *testing rounds.*

Proof Let us consider the worst case. In each round, assuming no attackers are identified and filtered out previously, we have $t(d, n') + \lceil \frac{n-n'}{\mathbf{w}} \rceil = K$. Since $t(d, n') \leq n'$ then $n' \geq \frac{Kw-w-n}{w-1}$. Therefore the maximum number of testing rounds needed is: $\frac{dn}{K} / \frac{Kw-w-n}{w-1} \leq dw^2/(Kw - w - n) = O(dw^2/(Kw - w - n))$. □

This complexity for PND is not as good as that of the previous two algorithms, partially due to the simple bound used. Therefore, we continue the analysis using the tight lower bound for *t*(*d*,*n*) proposed in [13] and investigate the corresponding performance of PND as follows.

Lemma 2.6 (*D'yachkov-Rykov lower bound*) [13] *Let* $n > w \geq d \geq 2$ *and* $t > 1$ *be integers. For any superimposed* $(d\text{-}1, n, w)$*-code* ((*d, n, w*)*-design*) *X of length t* (*X is called a* $(d\text{-}1)$*-disjunct matrix with t rows and n columns*), *the following inequality holds*:

$$t \geq \lceil \frac{dn}{w} \rceil$$

The following are the related performance analysis based on this lower bound of *t*(*d*,*n*).

Corollary 2.1 *In the* PND *algorithm, given* $w \geq d \geq 1$, *we have:* $n' = \min\{\frac{|A|w}{d+1}, n - Kw + w + |A|w\}$

proof According to Lemma 2.6, with number of columns $n'' \in [1, |S|]$, we have

$$|A| \geq \lceil \frac{(d+1)n''}{w} \rceil \geq \frac{(d+1)n''}{w} \Rightarrow n' = \max\ n'' \leq \frac{|A|w}{d+1}$$

Meanwhile, in the *PND* algorithm, for each round *i*, we have:

$$|A_i| \geq K - \lceil \frac{n-n'}{w} \rceil \Rightarrow n' \leq n - Kw + w + |A|w$$

□

Lemma 2.7 *In any testing round i*:

1. $n' = \frac{|A_\text{i}|w}{d+1}$ when $K \in \left(d, \frac{dw+n+w}{w}\right]$;
2. $n' = n - Kw + w + |A_\text{i}|w$ when $K \in \left[k_1, \frac{dn+n+w}{w}\right)$ with

$$k_1 = \frac{dw + w + n + \sqrt{(dw+w+n)^2 + 4wnd^2}}{2w}$$

Proof According to Corollary 2.1, we have:

1. We have

$$K \leq \frac{dw+n+w}{w} \Leftrightarrow wK \leq dw + n + w$$

$$|A| \geq d + 1 \Leftrightarrow \frac{|A|wd}{d+1} \geq wd$$

Hence

$$n + w - Kw + |A|w \geq \frac{|A|w}{d+1}$$

2. In order to get

$$\frac{|A|w}{d+1} \geq n - Kw + w + |A|w$$

we need

$$|A| \leq \frac{((K-1)m-n)(d+1)}{wd}$$

which requires

$$K - \frac{(K-d)n}{Kw} \leq \frac{((K-1)m-n)(d+1)}{wd}$$

Hence we need

$$wK^2 - (dw + w + n)K - nd^2 \geq 0$$

Solving this inequality, we have $K \in [k_1, +\infty)$. Note if $K \geq \lceil \frac{dn+n}{w} \rceil$, we need not do partitions in *PND* algorithm and since

$$k_1 \leq \frac{dn + n + w}{w}$$

we have

$$K \in [k_1, \frac{dn + n + w}{w})$$

Moreover,

$$k_1 > \frac{dw + w + n}{w} \geq \frac{d^2 + w + n}{w}$$

there are thus no overlaps between these two intervals. □

Therefore, we split $K \in (d, +\infty)$ into four disjoint intervals and study which interval of value K yields $O(1)$ testing rounds in worst case besides the interval $K \in [\frac{dn+n+w}{w}, +\infty)$ shown above, as well as complexity for other intervals.

- **I**: $K \in (d, \frac{dw+n+w}{w}]$ yields $n' = \frac{|A|w}{d+1}$;
- **II**: $K \in (\frac{dw+n+w}{w}, k_1)$ yields $n' = \min\{\frac{|A|w}{d+1}, n - Kw + w + |A|w\}$;
- **III**: $K \in [k_1, \frac{dn+n+w}{w})$ yields $n' = n - Kw + w + |A|w$;
- **IV**: $K \in [\frac{dn+n+w}{w}, +\infty)$ yields ONE testing round in total.

Lemma 2.8 *The* PND *algorithm needs at most* $O(1)$ *testing rounds with* $K \in [k_2, \frac{dw+n+w}{w}]$, *where*

$$d \leq \frac{w + \sqrt{w^2 - 4n^2(n - w)}}{2(n - w)}$$

and

$$k_2 = \frac{n + w + \sqrt{n^2 + w^2 + 2nw - 4n^2w + 4d^2wn + 4dwn}}{2w}$$

Proof Since at least one server gets positive outcome at the first testing round, we have

$$|A_0| \geq K - \lceil \frac{(K - 1)n}{Kw} \rceil$$

With simple algebraic computations, we can reach the interval $\left[k_2, \frac{dw+n+w}{w}\right]$ on the condition that

$$d \leq \frac{w + \sqrt{w^2 - 4n^2(n - w)}}{2(n - w)}$$

within interval **I**; however, for intervals **II** and **III**, no such detailed subintervals of K yielding $O(1)$ testing rounds can be obtained. □

Lemma 2.9 *Within interval* **I**, PND *algorithm can identify all* **ID***s with* $O(d + \frac{K}{\sqrt{n}})$ *testing rounds.*

Proof We derive the time complexity from the following recurrence:
Starting round 0: $|S_0| \le \frac{dn}{K}$, and $K - \lceil \frac{(K-1)n}{Kw} \rceil \le |A_0| \le K - \lceil \frac{(K-d)n}{Kw} \rceil$
Ending round T: $0 < |S_T| \le \frac{|A_\mathrm{T}|w}{d+1}$
Iteration: For $\forall i \in [0, T-1]$ we have

$$|S_{\mathrm{i+1}}| = |S_\mathrm{i}| - \frac{|A_\mathrm{i}|w}{d+1}$$

and

$$K - \lceil \frac{n - |S_{\mathrm{i+1}}|}{w} \rceil \le |A_{\mathrm{i+1}}| \le K - \lceil \frac{n - |S_{\mathrm{i+1}}| - d}{w} \rceil$$

hence

$$\begin{cases} |A_{\mathrm{i+1}}| \ge K - \frac{n}{w} + \frac{|S_\mathrm{i}|}{w} - \frac{|A_\mathrm{i}|}{d+1} - 1 \\ S_0 \le \frac{\sum_{i=0}^{T} |A_\mathrm{i}|w}{d+1} \end{cases}$$

In order to estimate the maximum time cost, use $|S_0| = \frac{dn}{K}$ to initiate the worst starting case. Solving this recurrence, we get the following inequality:
$\frac{Kw}{2(d+1)}T^2 - (\frac{Kw}{2(d+1)} + \frac{dn}{K} - K + \frac{n}{w} + w)T - (K - \frac{(K-1)n}{w} - \frac{dn(d+2)}{K} + w - 1) \le 0$,
therefore

$$T \le \frac{d+1}{Kw}\left(\alpha + \sqrt{\frac{\beta}{d+1} + \alpha^2}\right)$$

where

$$\alpha = \frac{Kw}{2(d+1)} + \frac{dn}{K} + \frac{n}{w} - K + w$$

and

$$\beta = 2K^2w - 2K^2n + 2Kn - 2Kw + 2Kw^2 - 2d(d+2)nw$$

Since $\frac{n}{w} \le K,\ wK \le dw + n + w$ and $n > w$, we have $\alpha > 0$ and $\beta > 0$. So with trivial computation we can get

$$\begin{aligned} T &\le \frac{2\alpha(d+1)}{Kw} + \sqrt{\frac{\beta}{d+1}} \\ &< 1 + \frac{2(d+1)^2}{K} + \sqrt{\frac{(4K-2)(d+1)}{n} + \frac{2(d+1)}{d+1}} \\ &< 1 + 2(d+1) + \sqrt{\frac{4K(d+1)}{n} + 2} \\ &< 3 + \sqrt{2} + 2d + \frac{2K}{\sqrt{n}} \end{aligned}$$

Therefore, *PND* will complete the identification within at most $O(d + \frac{K}{\sqrt{n}})$ testing rounds. □

Note that since K is always much smaller than n, the complexity will approach to O(d) in fact.

Lemma 2.10 *Within interval* **III**, *PND can identify all* **ID***s with at most O(d) testing rounds.*

Proof Similarly, we can get $T \leq 2d+1+2\sqrt{\frac{1+d}{4}}$ by solving the following recurrence
Starting round 0: $|S_0| = \frac{dn}{K}$ and $K - \lceil\frac{(K-1)n}{Kw}\rceil \leq |A_0| \leq K - \lceil\frac{(K-d)n}{Kw}\rceil$
Ending round T: $0 < |S_{\mathrm{T}}| \leq n - Kw + w + |A_{\mathrm{T}}|w$
Iteration: $\forall i \in [0, T-1],\ |S_{\mathrm{i+1}}| = |S_{\mathrm{i}}| - (n - Kw + w + |A_{\mathrm{i}}|w)$ and $K - \lceil\frac{n-|S_{\mathrm{i+1}}|}{w}\rceil \leq |A_{\mathrm{i+1}}| \leq K - \lceil\frac{n-|S_{\mathrm{i+1}}|-d}{w}\rceil$. Hence, *PND* will complete the identification within at most O(d) testing rounds. □

Corollary 2.2 *Within interval* **II**, PND *algorithm can identify all* **ID***s with* $O(d+\frac{K}{\sqrt{n}})$ *testing rounds.*

Proof According to Lemmas 2.9 and 2.10, the time complexity of *PND* algorithm depends on the value of n' at each round, and since n' within interval **II** oscillates between $\frac{|A|w}{d+1}$ and $n - Kw + w + |A|w$, the time complexity is at most $O(d + \frac{K}{\sqrt{n}})$. □

Theorem 2.3 *Given* $1 \leq d \leq w$, *the PND algorithm can identify all* **ID***s within*

1. *at most* $O(d + \frac{K}{\sqrt{n}})$ *testing rounds when* $K \in (d, k_1)$, *whilst at most* $O(1)$*testing rounds when* $K \in [k_2, \frac{dw+n+w}{w}]$ *on condition that* $d \leq \frac{w+\sqrt{w^2-4n^2(n-w)}}{2(n-w)}$;
2. *at most* $O(d)$ *testing rounds when* $K \in [k_1, \frac{dn+n+w}{w})$;
3. *at most* $O(1)$ *testing rounds when* $K \in [\frac{dn+n+w}{w}, +\infty)$; *where*
   $k_1 = \frac{dw+w+n+\sqrt{(dw+w+n)^2+4wnd^2}}{2w}$
   and $k_2 = \frac{n+w+\sqrt{n^2+w^2+2nw-4n^2w+4d^2wn+4dwn}}{2w}$.

Despite the number of needed testing rounds differing for the three algorithms above, the time complexity of calculating each testing round for these algorithms is approximate in practice. It is trivial to see that the costs for *SDP* and *SDoP* are negligible, but not for *PND* algorithm which involves polynomial computation on Galois Field. However, considering that the upper bound of both the number of clients n and attackers d are estimated, the detection system can precompute the d-disjunct matrices for all possible (n,d) pairs offline, and fetch the results in real-time. Therefore, the overhead can be decreased to $O(1)$ and the client requests can be smoothly distributed at the turn of testing rounds without suffering from long delays of matrix update. In some cases, if an online construction of M is required, we can use a randomized construction method which is presented in Chap. 4. This method has a very low computational overhead.

2.5 Detection System Configuration

In this section, we present the rest of the system configuration which illustrates how to implement the above mathematical framework into a practical solution for an application DoS defense . More specifically, we discuss *how to distribute client requests based on M with a low overhead* and *how to generate test outcome with high accuracy*.

2.5.1 System Overview

As mentioned in the GT-based detection model, each back-end server works as an independent testing domain, where all virtual servers within it serve as testing pools. In the following sections, we only discuss the operations within one back-end server, and it is similar in all other servers. The detection consists of multiple testing rounds, and each round can be sketched in four stages (Fig. 2.4):

1. **Stage 1**: Matrix M for testing is generated and updated.
2. **Stage 2**: All clients are distributed to virtual servers based on M. The back-end server maps each client into one distinct column in M and distributes an encrypted token queue to it. Each token in the token queue corresponds to a 1-entry in the mapped column. i.e., client j receives a token with destination virtual server i *iff* $M[i, j] = 1$. Being piggybacked with one token, each request is forwarded to a virtual server by the virtual switch. In addition, requests are validated on arriving at the physical servers for faked tokens or identified malice **ID**. This procedure ensures that all the client requests are distributed exactly as how the matrix M regulates, and prevents any attackers from accessing the virtual servers other than the ones assigned to them.
3. **Stage 3**: All the servers are monitored for their service resource usage periodically. Specifically, the arriving request aggregate (the total number of incoming requests) and average response time of each virtual server are recorded and compared with some dynamic thresholds (to be shown later). All virtual servers are associated with positive or negative outcomes accordingly.
4. **Stage 4**: Based on these testing outcomes and M, decode and identify legitimate or malicious **ID**s. By following the detection algorithms discussed above, all the attackers can be identified within several testing rounds.

To lower the overhead and delay introduced by the mapping and piggybacking for each request, the system is exempted from this procedure in normal service state. As shown in Fig. 2.5, the back-end server cycles between two states, which we refer to as NORMAL mode and DANGER mode. Once the estimated response time (ERT) of any virtual server exceeds some profile-based threshold, the whole back-end server will transfer to the DANGER mode and execute the detection scheme. Whenever the

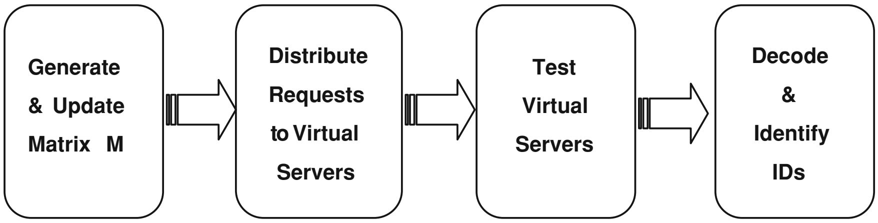
Generate & Update Matrix M
Distribute Requests to Virtual Servers
Test Virtual Servers
Decode & Identify IDs

Fig. 2.4 The four stages within one testing round in DANGER mode

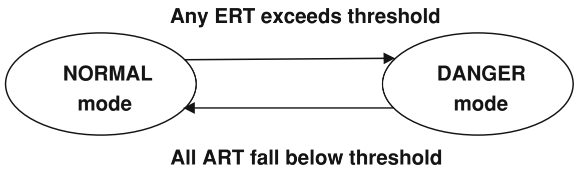

Fig. 2.5 Two-state diagram of the system. Once the estimated response time (ERT) of any virtual server exceeds some profile-based threshold, the whole back-end server will transfer to the DANGER mode and execute the detection scheme. Whenever the average response time (ART) of each virtual server falls below the threshold, the physical server returns to NORMAL mode

average response time (ART) of each virtual server falls below the threshold, the physical server returns to NORMAL mode.

2.5.2 Configuration Details

Distributing Tokens. Two main purposes of utilizing tokens are associating each client with a unique, non-spoofed **ID** and assigning them to a set of virtual servers based on the testing matrix. On receiving the connection request from a client, each back-end server responses with a token queue where each token is of 4-tuple: (client ID, virtual server ID, matrix version, encrypted key). "client ID" refers to the unique non-spoofed **ID** for each client, which we assume unchanged during the testing period (DANGER mode). "virtual server ID" is the index of each virtual server within the back-end server. This can be implemented as a simply index value, or through a mapping from the IP addresses of all virtual servers. The back-end server blocks out-of-date tokens by checking their "matrix version" value, to avoid messing up the request distribution with non-uniform matrices. With regard to the "encrypted key", it is an encrypted value generated by hashing the former three values and a secured service key. This helps rule out any faked tokens generated by attackers.

Length of Testing Round. Since we need to distribute the requests exactly the way M regulates, it is possible that: if P is too short, some clients may not have distributed their requests to all their assigned servers, i.e., not all the 1-entries in M are matching with at least one request. For an attacker who launches a low-rate high-workload attack, its high-workload requests may only enter a part of its assigned servers, so false negative occurs in this case. However, if P is too long, the detection latency will be significantly increased. Therefore, P must be carefully decided to be just long enough for each client to spread their requests to all the assigned servers, i.e., each column needs to be mapped with at least $\sum_{i=1}^{t} M[i, j]$ requests. Thus, we have:

$$P = \max_{j=1}^{n} \sum_{i=1}^{t} M[i, j]/r_{\min}$$

where $r_{\min}$ denotes the minimum interarrival request rate, provides a theoretical lower bound of P.

Legitimate Profile. The legitimate profile does not refer to the session behavior profile for legitimate clients, but instead records the distribution of the *ART* on a virtual server receiving only legitimate traffic. Malicious requests are certainly to generate destructions to the victim server machines, whose *ART* will usually be much higher than that of normal cases. Therefore, *ART* can work as an indicator of the application resource usage. However, the resource usage varies for different time intervals (peak/non-peak time) due to the change of client quantity, so we also investigate the *ART* distributions regarding each possible number of clients, assuming that there are at most n clients.

A sample construction of this profile is: The distributions of *ART* in legitimate traffic at different time intervals for several weeks are obtained after the system is established. Legitimate traffic can be achieved by de-noising measures and ruling out the influences of potential attacks [14]. The *ith* entry of the profile records the distribution of *ART* on a virtual server with $i \in [1, n]$ clients. Specifically, we assign i clients to a virtual server, and evenly divide the round length P into 100 subintervals (which we refer as P_{sub} throughout the paper), and monitor the server *ART* within each subinterval P_{sub}.

NORMAL mode and Transfer Threshold. To decrease the length of detection period, where additional state maintenance overhead and service delay cannot be avoided, the back-end server cluster provides normal service to clients, without any special regulations. This is referred as NORMAL mode, which takes *ERT* (estimated response time) as a monitoring object. Note that *ERT* is an expected value for *ART* in the near future, and can be computed as:

$$ERT = (1 - \alpha) \cdot ERT + \alpha \cdot ART$$

where α is decay weight for smoothing fluctuations. Since the interarrival rate and workload of client request can be randomized distributed, it is difficult to perfectly fit the *ART* distribution using classic distribution functions. Considering that Normal Distribution [15] can provide an approximate fitting to *ART*, a simplified threshold can be adpoted based on the Empirical Rule as follows: If any virtual server has an $ERT > \mu + 4\sigma$ (μ and σ denote the expected value and standard deviation of the fitted *ART* distribution), the back-end server is probably undergoing an attack, and thus transfers to DANGER mode for detection. To enhance the accuracy of this threshold, more sophisticated distributions can be employed for fitting the samples, however, higher computation overhead will be introduced for the threshold.

DANGER mode and Attack Threshold. In DANGER mode, besides the *ART* values, the back-end server simultaneously counts the arriving request aggregate within each P_{sub} for each virtual server. The motivation of this strategy arises from the fact that, high-rate DoS attacks always saturate the server buffer with a flood of malicious requests. By counting the number of arrivings in the buffer requests periodically, possible high-rate attacks can be detected even before depleting the service resources.

We predefine R as a maximum legitimate interarrival rate, and derive an upper bound of the arriving request aggregate C_{j} for virtual server j within period P_{sub} as:

$$C_{\text{j}} = \sum M[j,i] \cdot \lfloor \frac{\sum_{s=j+1}^{t} M[s,i] + R \cdot P_{\text{sub}}}{\sum_{k=1}^{t} M[k,i]} \rfloor$$

Once this threshold is violated in a virtual server, it undergoes high-rate attacks or flash-crowd traffic. A positive outcome can be generated for this testing round.

The outcome generation by monitoring the *ART* value for a virtual server consists of the following detailed steps:

1. Check the *ART* distribution profile for the values of parameters μ and σ, as mentioned in the NORMAL mode section;
2. among all the 100 subintervals P_{sub} (each is $P/100$) within the current testing period P, if no violation: $ART \geq \mu + 4\sigma$ occurs in any P_{sub}, the virtual server gets negative outcome for this testing round;
3. if for some subinterval P_{sub}s, $ART \geq \mu + 4\sigma$ occurs, this virtual server is in danger, either under attack, or undergoing flash-crowd traffic. In this case, we wait till the end of this round to get the distribution of *ART* values for all P_{sub} s for further decision;
4. if the ratio of "danger" subintervals P_{sub} s with $ART \geq \mu + 4\sigma$, over the total subinterval amount (100 in this case), exceeds some quantile regulated by the *Empirical Rule*, e.g., quantile 4% for confidence interval $[\mu + 2\sigma, \infty)$, this virtual server will be labeled as positive;
5. for the other cases, the virtual server will have a negative outcome.

After identifying up to d attackers, the system remains in the DANGER mode and continues monitoring the *ART* for each virtual server for one more round. If all the virtual servers have negative outcomes, this back-end server is diagnosed as healthy and return to the NORMAL mode, otherwise further detections will be executed (because of error tests).

With regard to the two objects monitored, C_{j} provides different counting thresholds for different virtual servers, while *ART* profile supplies dynamic thresholds for virtual server containing different amounts of client. The combination of these two dynamic thresholds helps decrease the testing latency and false rate.

2.6 Experimental Analysis

To demonstrate the theoretical complexity results shown in the previous section, we present here a simulation study which has been conducted by [16] in terms of four metrics: *average testing latency* T which refers to the length of the time interval from attackers starting sending requests till all of them are identified; *average false positive rate* f_p and *false negative rate* f_n; as well as the *average number of testing rounds* R_{test} which stands for the number of testing rounds needed for identifying all the clients by each algorithm.

2.6.1 Configurations

To this end, a simulator in Java by modeling both the n clients and K virtual servers as independent threads was implemented. In order to mimic the real client (including attacker) behavior and dynamic network environment, the client/server system was implemented as follows:

- each legitimate client joins in and leaves the system at random times which are uniformly distributed, while the attacker threads arrive at time $t = 30$ s and keep live until being filtered out.
- both legitimate and malicious clients send requests with a random interarrival rate and CPU processing time (workload) to the virtual servers, however, legitimate ones have a much smaller random range than that of the attackers.
- each virtual server is equipped with an infinite request buffer and all the client requests arrive at the buffers with 0 transmission and distribution delays, as well as 1 ms access time for retrieving states from the shared memory; each server handles the incoming requests in its own buffer in FCFS manner and responds to the client on completing the corresponding request; the average response time and incoming request aggregate are recorded periodically to generate the test outcomes by comparing them to the dynamic thresholds fetched from established legitimate profiles.

The purpose of assuming both transmission and distribution delays to be 0 is to simply quantify the length of the whole detection phase (testing latency). With regard to the transmission delay, it can be large due to the geographical distances in large-scale distributed systems and possibly can bring up the testing latency if the client sends request in a stop-and-wait manner (it does not send a request until the previous requests are all responded, therefore the request rate is quite low and the length of each testing round is required to be longer), yet since the detection will be completed just in several rounds, such increases in the detection length are not significant. The assumption of 0 distribution delay is also practical, since the computational overheads for the testing matrix and dynamic thresholds are negligible by precomputing and fetching the results from the profiles. For the 1 ms state maintenance time, since all

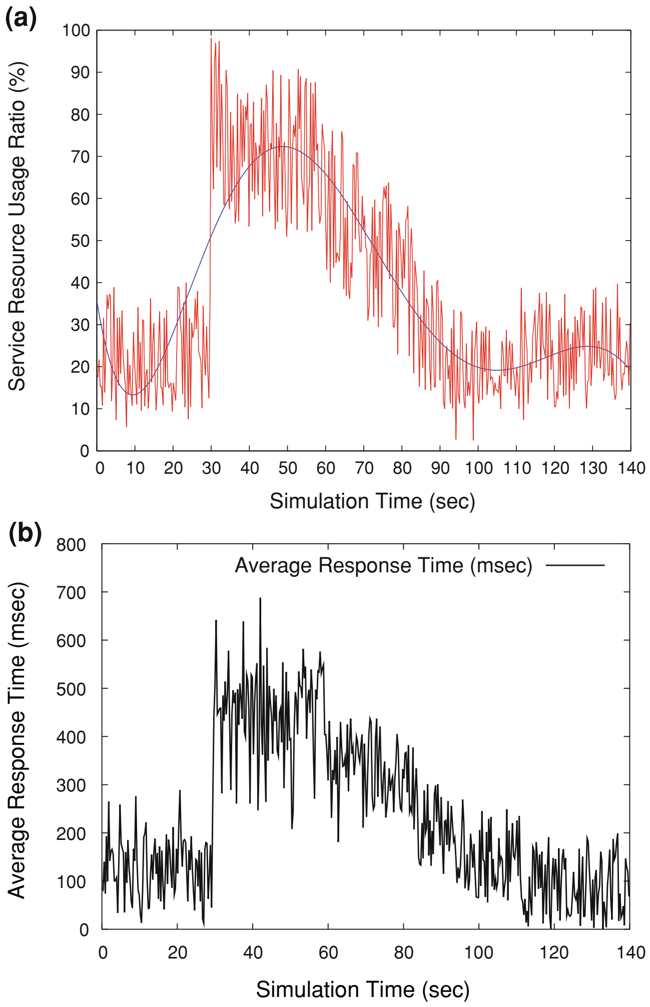

Fig. 2.6 Status of the back-end server

the clients are residing in one physical server, and all the virtual servers can quickly retrieve the client states from the shared memory, this is also a practical assumption.

With regard to the details of client behavior, the legitimate request interarrival rate is randomized from 1 to 3 request per ms, and the legitimate workload is randomized from 1 to 3 ms CPU processing time. On the contrary, the malicious request interarrival rates range from 5 to 20 per ms, and malicious workload range from 5 to 20 ms CPU time. Although requests with arbitrarily large rate or workload are favored by attackers, they are in fact easier to be discovered, so malicious requests with small margin from legitimate ones were also considered.

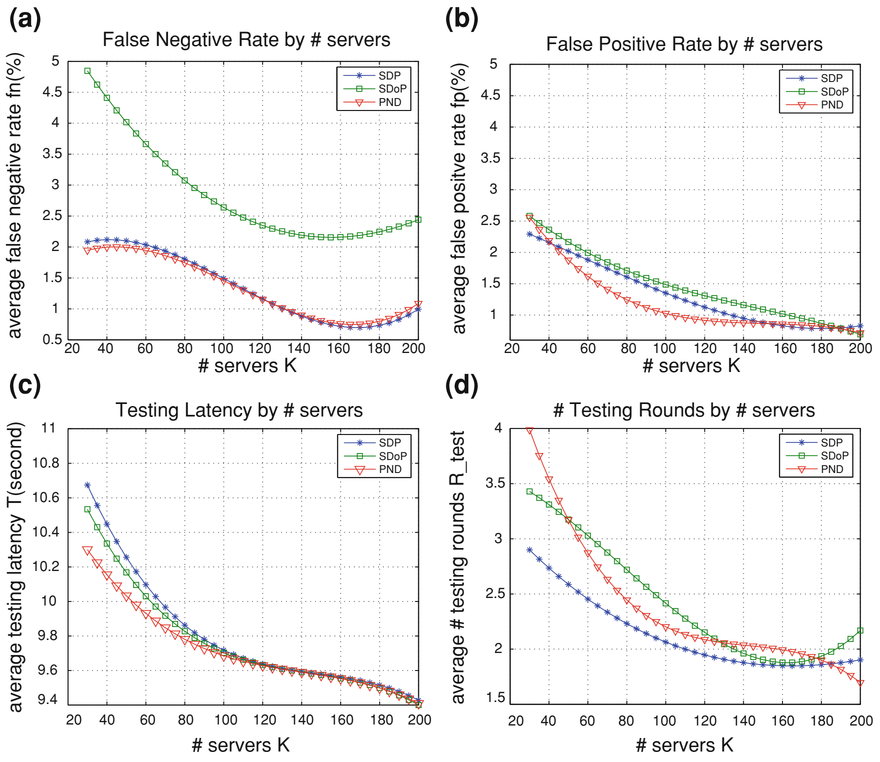

Fig. 2.7 Robustness by different numbers of back-end server machines k

2.6.2 Results

By setting $K = 50, n = 1,000, w = 100, d = 40, P = 1$ second, the efficiency of the GT-based detection system using *PND* algorithm, in terms of reducing the service resource usage ratio (*SRUR*) and the average response time (*ART*) are shown in Fig. 2.6a and b. The values of *SRUR* and *ART* climb up sharply at $t = 30$ s when the attack starts, and then gradually falls to normal before $t = 100$ s. Therefore, it takes only 70 s for the system to filter out attackers and recover to normal status. Note that actual detection latency should be shorter than this, because the threshold of *ART* for the system to convert from DANGER mode back to NORMAL mode is slightly higher than normal *ART*. Therefore, the system *SRUR* and *ART* will recover to normal shortly after the detection period ends.

In the following we show the robustness of the performance toward different environment settings: an increasing number of (1) virtual servers K; (2) malicious clients d; (3) all clients n.

In Fig. 2.7, a simulation of identifying $d = 10$ attackers out of $n = 1,000$ clients with the number of virtual servers ranging in [17, 200] was studied. As can be seen, on one

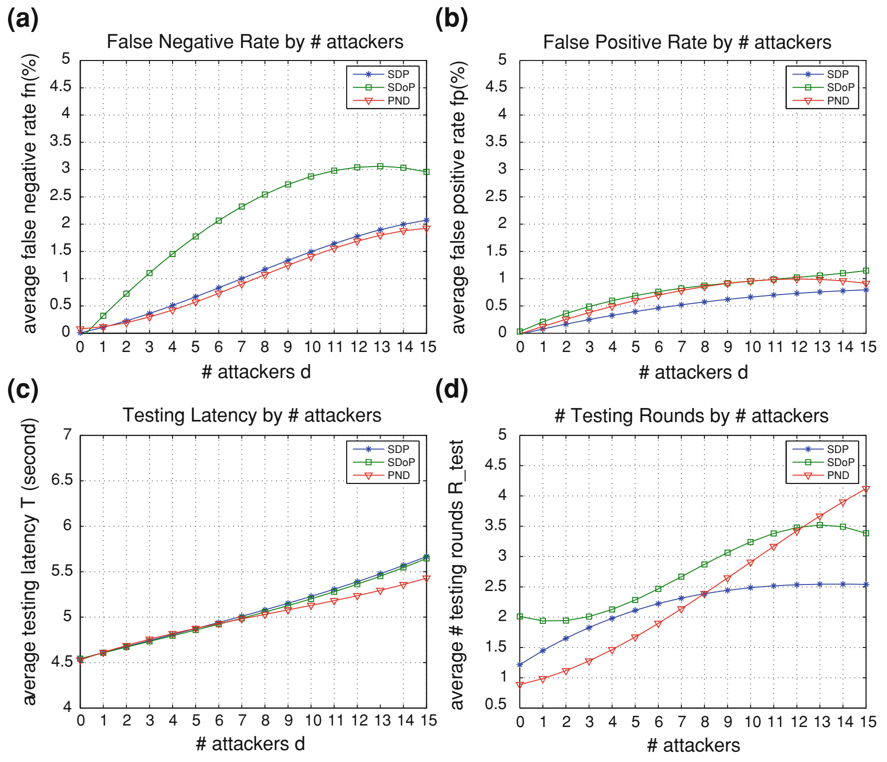

Fig. 2.8 Robustness by different numbers of attackers *d*

hand, all the false negative (Fig. 2.7a) and positive (Fig. 2.7b) rates are upper bound by 5% and decreasing as *K* goes up for all the three algorithms. This makes sense since the most possible case for an attacker to succeed in hiding itself is that it is accompanied by many clients with low request rate and workloads in a testing pool. In this case, it is quite possible that the aggregate interarrival rates and workloads in this server are still less than that of a server with the same number of legitimate clients. Therefore, the less clients serviced by each server, the less possibly this false negative case happens. Note that even if there is only one attacker but no legitimate client in a server whose incoming traffic is not quite dense, the server can still be tested as positive since all the tests are conducted based on some dynamic threshold (use the threshold for server with one client in this case), which changes as the number of active clients varies in the current virtual server. On the other hand, the testing latencies and the number of testing rounds keep declining from less than 11 s and 4 rounds respectively, which is because the identification will speed up with more available virtual servers.

With respect to the three different algorithms, they obtained approximate performances except that *SDoP* has slightly higher false negative rate than the other two.

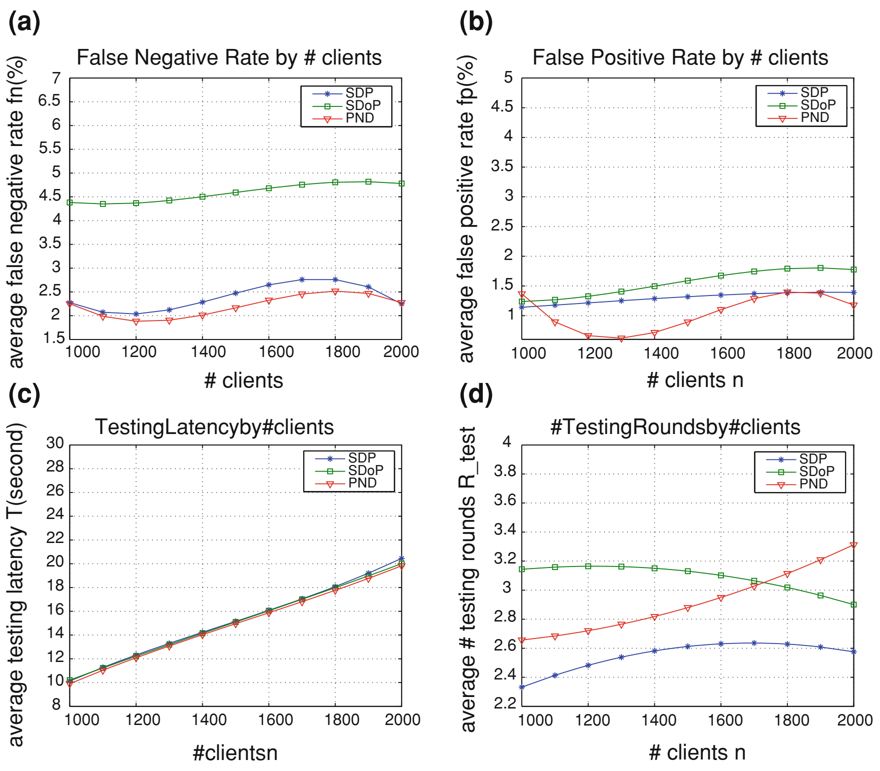

Fig. 2.9 Robustness by different total numbers of clients n

This is because of the legitimate clients identified at the earlier stages and staying in the servers till the end of the detection. Since their request rates and workloads are likely to be smaller than normal (that is why they are identified earlier), they may camouflage attackers in the following rounds.

In Fig. 2.8, the identification for $n = 1,000$, $K = 50$, $w = 100$, $P = 1$ s with [3, 17] attackers was conducted. As illustrated in Fig. 2.8, all the values of the four measures are slowly increased as d goes up. Overall, the false negative/positive rates are limited to less than 5% and the testing latencies are smaller than 6 s with 5 testing rounds. Similar to the previous experiments, *PND* and *SDP* exhibit better performances than *SDoP* in terms of false negative rate, but for the other measures, they are approximately the same.

Figure 2.9 shows its robustness for the cases $d = 10$, $K = 50$, $w = 100$, $P = 1$ s with [1,000, 2,000] clients. Apparently, both the false rates and the number of testing rounds keep stably below 5 % and 4 rounds respectively, toward increasing client amount. The testing latencies grow up from 10 to 20 s for all three algorithms, due to the increasing time costs for state maintenance toward a double number of clients (from 1,000 to 2,000). However, this small latency is still tolerable in real-time applications and can be further reduced by decreasing the state maintenance time cost within the same physical machine.

Overall, the simulation results can be concluded as follows:

- in general, the system can efficiently detect the attacks, filter out the malicious clients, and recover to NORMAL mode within a short period of time, in real-time network scenarios;
- all the three detection algorithms can complete the detection with short latency (less than 30 s) and low false negative/positive rate (both less than 5%) for up to 2,000 clients. Thus they are applicable to large-scale time/error-sensitive services;
- the *PND* and *SDP* algorithms achieve slightly better performance than the *SDoP* algorithm. Furthermore, the efficiency of the *PND* algorithm can be further enhanced by optimizing the *d*-disjunct matrix employed and state maintenance efficiency.

References

1. Sekar V, Duffield N, van der Merwe K, Spatscheck O, Zhang. H (2006) LADS: large-scale automated DDoS detection system. In: USENIX annual technical conference 2006
2. Kandula S, Katabi D, Jacob M, Berger AW (2005) Botz-4-sale: surviving organized DDoS attacks that mimic flash crowds 2nd NSDI. MA, Boston, May 2005
3. Ranjan S, Swaminathan R, Uysal M, Knightly E (2006) DDos-resilient scheduling to counter application layer attacks under imperfect detection. In Proceedings of the IEEE infocom, barcelona, Spain, April, 2006
4. Kim Y, Lau WC, Chuah MC, Chao HJ (2004) Packetscore: statisticsbased overload control against distributed denial-of-service attacks. In: Proceedings of infocom, HongKong, 2004
5. Du DZ, Hwang FK (2006) Pooling designs: group testing in molecular biology. World Scientific, Singapore
6. Atallah MJ, Goodrich MT, Tamassia R (2005) Indexing information for data forensics, ACNS. Lecture notes in computer science vol 3531. Springer, Heidelberg, pp 206–221
7. Ricciulli L, Lincoln P, Kakkar P (1999) TCP SYN flooding defense. In: Proceedings of CNDS
8. Gligor VD (2003) Guaranteeing access in spite of distributed service-flooding attacks. In: Proceedings of the security protocols workshop
9. Kargl F, Maier J, Weber M (2001) Protecting web servers from distributed denial of service attacks. In WWW '01: Proceedings of the 10th international conference on World Wide Web. ACM Press, New York, USA, pp 514–524
10. Thai MT, Xuan Y, Shin I, Znati T (2008) On detection of malicious users using group testing techniques. In: Proceedings of IEEE international conference on distributed computing systems (ICDCS)
11. Sharma P, Shah P, Bhattacharya S (2003) Mirror hopping approach for selective denial of service prevention in WORDS'03
12. Service provider infrastructure security: detecting, tracing, and mitigating network-wide anomalies (2005). http://www.arbornetworks.com 2005
13. Chu Y, Ke J (2007) Mean response time for a G/G/1 queueing system: simulated computation. Appl Math Comput 186(1):772–779
14. Eppstein D, Goodrich MT, Hirschberg D (2005) Improved combinatorial group testing algorithms for real-world problem sizes WADS. LNCS vol 3608. Springer, Heidelberg, pp 86–98

15. Mori G, Malik J (2003) Recognizing objects in adversarial clutter: breaking a visual captcha. IEEE Computer Vision and Pattern Recognition
16. Dyachkov AD, Rykov VV, Rachad AM (1989) Superimposed distance codes. Prob Control Inform Thy 18:237–250
17. Dyachkov AG, Macula AJ, Torney DC, Vilenkin PA (2001) Two models of nonadaptive group testing for designing screening experiments. In: Proceeding 6th International workshop on model-oriented designs and analysis. p 635