On Detection of Malicious Users Using Group Testing Techniques

My T. Thai Ying Xuan Incheol Shin
Dept of Comp and Info Sci and Eng
University of Florida. Gainesville, FL 32611
{mythai, yxuan, ishin}@cise.ufl.edu

Taieb Znati
Computer Science Department
University of Pittsburgh. Pittsburgh, PA 15215
znati@cs.pitt.edu

Abstract

Despite decades of research, there have not been developed concrete defense solutions for most of current attacks to Internet services, let alone new attack types. An essential problem to overcome is that malicious traffic can be similar to legitimate ones. Thus a more fundamental model which should be based on the overall performance of servers/subnets without inspecting each traffic must be remedied. Based on this observation, we propose a novel system framework, called Detection of Malicious Users (DMU) which attempts to solve various attack types. Motivated by DMU, we introduce a new theoretical model, called Size Constraint Group Testing (SCGT). Several algorithms based on SCGT for various networking scenarios are proposed. We also provide several fundamental results on SCGT, revealing some necessary conditions to obtain an $O(1)$ detection time algorithm.

1. Introduction

Internet security is increasing in importance. Yet, despite decades of research, we are still unable to make secure computer networks. Further, more sophisticated and new attacks are expected to continue posing a greater degree of threat to Internet services. As a result, a more fundamental model, in terms of theoretical and system perspectives, regardless of attack types must be investigated.

An essential problem to overcome for any defense mechanism is the fact that malicious traffic/packets can be similar to legitimate ones. For example, let us consider the Distributed Denial of Service (DDoS), which is the second largest contributor to all financial losses due to cybercrime [3]. Since its first appearance, there have been several proposed models but no fundamental defense solution has been developed yet [1, 2]. The current solutions either rely on some assumptions on traffic flow/rate to directly determine malicious traffic or bases on the difference between current traffic and predicted normal traffic [2]. However, attackers can change their attack patterns to overthrow the assumption and evade detection. And it is difficult or impossible for the training data to provide all types of normal traffic behavior, thus legitimate traffic can be classified as attack traffic. Moreover, attackers can use “legitimate traffic” generators to avoid detection. Therefore, a defense mechanism solely based on the differences in traffic and packets to directly identify and filter malicious traffic will generate high false positive and not be very effective.

A main observation is that despite existing or new attack methods, the most common goal is to degrade or paralyze the performance of servers/subnets. Thus monitoring the overall performance of servers/subnets will indicate whether a server is under attack but would not specify which traffic is malicious. Beside the detection time and false positive rate, the biggest challenges of this method are two-fold: (1) How to differentiate malicious traffic from legitimate ones using only the overall performance without inspecting traffic on the individual. (2) How to mitigate the attack so that servers are still able to serve legitimate traffic during the detection period.

In this paper, we propose a fundamental model, called Detection of Malicious Users (DMU), and its mathematically rigorous model, called Size Constraint Group Testing (SCGT). The DMU model aims to offer a universal defense solution for various attack types based on group testing techniques, so that we can detect and eliminate the malicious traffic in real-time without interrupting legitimate services. If attackers do not spoof the source addresses of compromised agents (which is true in some DDoS attacks [2]), DMU can also identify malicious agents in real-time. In addition, DMU can effectively mitigate the attacks and quickly migrate legitimate services by dynamically moving users to different servers and balancing the server load. Note that the term “users” can be referred as traffic, flows, connections, or users themselves. Each individual can be separated by its source address.

The realization of DMU, however, raises several chal-
challenges: (1) What reassignment patterns must be used to identify all malicious users in a short period of time? (2) How the reassignment can be achieved with minimum expenses? (3) How the legitimate services can be automatically and seamlessly migrated to other servers during their connections? (4) What robust methods can be used to quickly identify servers under various attacks?

In this paper, we briefly present the DMU system architecture with a short discussion on points (2)-(4) due to space limitation. The details can be found in our extended version [15]. The rest of this paper is more focused on solutions of point (1), which is one of the most challenging in our model. To solve point (1), we use a combinatorial Group Testing (GT) technique, of which the idea is to pool items into a group, called pool for testing, instead of testing them one by one. When the outcome of a group test is negative, then all items in the pool are good. Otherwise, there exists at least one defective item (but do not know which one) in the pool and further testing on them are necessary. Various models based on GT have been investigated for many applications in several fields [4]-[10]. However, none of them has been used for security related topics.

Motivated by DMU, we introduce a new model, called Size Constraint Group Testing (SCGT). The main difference between this model and other existing GT models is that it has a size constraint on both the maximum number of items grouped together and the maximum number of pools used. With these size constraints, it poses several challenges in the GT design. Using SCGT, we propose three algorithms, namely Sequential Identification with Packing (SIP), Sequential Identification without Packing (SIoP), and Partial Non-adaptive Identification (PNI). These three algorithms consider various networking scenarios. The theoretical analysis shows that these algorithms successfully identify all malicious users in the short detection time. We also present several important properties underlying the SCGT model, which allow us to obtain an algorithm with $O(1)$ detection time within certain scenarios.

The rest of this paper is structured as follows. Section 2 provides the DMU system architecture and its SCGT mathematical model. The description and theoretical analysis of three proposed algorithms: SIP, SIoP, and PNI are presented in Section 3. Section 4 ends the paper with some discussions.

2. DMU Architecture and SCGT Model

2.1. DMU System Architecture

We will use virtual IP addresses (VIPAs) to control the user rearrangement on each server. Each user (based on its source address) will be associated with one distinct VIPA. This VIPA will be assigned to a server. Note that each server can have several VIPAs. The users will connect to servers through their VIPAs. During the connection, the user can be dynamically and transparently switched to different servers based on VIPA reassignments. For example, user $A$ has a VIPA $x$ which is assigned to server 1. At time slot 1, user $A$ connects to server 1 through VIPA $x$. At time slot 2, we reassign VIPA $x$ to server 4, thus user $A$ is now on server 4.

Conceptually, DMU has three main components as shown in Fig. 1: (1) **Firewall**: Used to inspect and regulate network flows entering our system. It may reject some malicious flows when they pass by [12], according to basic filtering rule. However, its main function is to drop all identified malicious flows after our testing procedures. (2) **Dynamic Address Assignment Handlers (DAAH)**: Responsible for assignment of VIPA for incoming flows and maintenance of forwarding tables. After a packet coming through the firewall, DAAH will assign VIPA to associate with a user by modifying the destination address and store these mappings in the DAAH servers. When a response from this packet comes back to the client, the source address of the response from the server will be restored by replacing the virtual IP with a public IP address. DAAH is also responsible for balancing the incoming packets to avoid being overloaded. (3) **Servers**: Consist of productive servers, which have two functions: normal services and real-time malicious detection. The performance of these servers is monitored to quickly detect any abnormal signals indicating that they are under attack using either SNMP-based monitoring device [14] or other methods [13]. Note that we just monitor the overall performance, not inspecting each packet/traffic.

The most challenging problem in this model is which rule this VIPAs assignment must follow. Let us first present some notations we will use throughout this paper:

- $n$: number of users, also number of Virtual IP Addresses (VIPAs)
- $K$: number of servers
- $d$: maximum number of malicious users, thus $1 \leq d \leq$
n. (Normally, \( d \ll n \)).

- \( w \): maximum number of users (or VIPAs) that can be assigned on each server, called server capacity.

The VIPA assignment is equivalent to this problem: Given \( K \) servers with server capacity \( w \) and \( n \) users randomly connecting to these servers. Assume there are at most \( d \ll n \) malicious users attacking these servers. The problem asks us to identify all malicious users in the shortest period of time using these \( K \) servers. We will use GT to solve this problem, which evokes a new GT model.

### 2.2. Size Constraint Group Testing Model

The traditional GT consisting of \( n \) items and \( t \) pools can be represented by a \( t \times n \) binary matrix \( M \) with row representing the pools and columns representing the items. An entry \( M[i,j] = 1 \) iff the \( i^{th} \) pool contain the \( j^{th} \) item; otherwise, \( M[i,j] = 0 \). Given this \( M_{t \times n} \) matrix, the test outcomes of these \( t \) pools can be represented by a \( t \)-dimensional column vector \( V \), called the test outcome vector. Note that \( V \) is a binary vector, in which \( 1 \) represents a positive outcome whereas \( 0 \) represents a negative outcome. For example, if \( V[i] = 0 \) then all items in pool (row) \( i \) are good; if \( V[i] = 1 \) then there exists at least one defected item in row \( i \).

Consider the matrix \( M \) and test outcome vector \( V \) (Eqn. (1)). Let \( w_i \) represent the number of 1-entry in row \( i \), that is \( w_i = \sum_{j=1}^{n} M[i,j] \). \( w_i \) is called the weight of row \( i \) in \( M \). Let \( w \) be an upper bound on the row weight and \( K \) is an upper bound on \( t \), the number of pool used. The SCGT model asks us to identify all \( d \) defected items in the minimum period of time by constructing a matrix \( M_{t \times n} \) such that \( w_i \leq w \) for all \( i \) and \( t \leq K \).

\[
M = \begin{pmatrix}
0 & 0 & 1 & \cdots & 0 \\
0 & 1 & 0 & \cdots & 0 \\
\vdots \\
0 & 1 & 0 & \cdots & 1
\end{pmatrix}
\]

\[
\Rightarrow V = \begin{pmatrix}
0 \\
1 \\
\vdots \\
0
\end{pmatrix}
\] (1)

In the context of VIPA assignments, the columns in \( M_{K \times n} \) represent the users (or VIPAs) and rows in \( M \) represent the servers. \( M[i,j] = 1 \) iff user (VIPA) \( j \) is activated on server \( i \). \( V[i] = 1 \) if server \( i \) is under attack, that is, there exists at least one malicious user currently activated on server \( i \). Otherwise, \( V[i] = 0 \). Note that the row weight is corresponding to the number of VIPAs which can be assigned on each server to control the server load. The problem asks us to assign all these VIPAs on at most \( K \) servers such that we can identify all malicious users in the shortest detection time. Note that during the detection period, we cannot disconnect the service of any legitimate user.

### 3. Three Proposed Algorithms and Analysis

In this section, we present three algorithms which solve the VIPA assignment problem in different network scenarios and provide their performance analysis in terms of detection time. In the analysis, we assume that each testing round costs \( O(1) \) testing time.

#### 3.1. Sequential Identification with Packing

In this section, we consider a networking scenario where each VIPA can be assigned to exactly one server. Therefore, each malicious user can attack only one server during each VIPA assignment.

**Algorithm Description.**

First, let us define some more notations used in the algorithm description. Let \( S \) be the set of unidentified users. (We will use “users” and “VIPAs” interchangeably in this paper since each VIPA associates uniquely to each user). Initially, \( S \) is a set of all \( n \) users. As soon as we identify the legitimate users, we will remove them from \( S \). Hence \( S \) consists of all suspect users. Let \( I \) be a set of servers under attack. The set of suspect users consists of all users currently residing on the attacked servers. Let \( A \) denote a set of available testing servers out of \( K \) given servers. Note that \( I \subseteq A \). And finally, let \( G \) denote the number of good servers for serving legitimate users. Thus \( G = K - |A| \).

**Algorithm 1 Sequential Identification with Packing**

1. \( S \leftarrow \) set of unidentified VIPAs; \( A \leftarrow \) set of available testing servers
2. \( w_i \leftarrow \) number of VIPAs on server \( i \); \( I \leftarrow \) set of server under attack
3. \( G \leftarrow \) number of servers for serving legitimate VIPAs
4. while \(|S| \neq 0\) do
5. for all server \( i \) in \( A \) do
6. \( w_i \leftarrow \left| \{ S \mid i \in I \cap A \} \right| \) \( \Rightarrow \) Randomly assign \( S \) to \( A \), no different servers have the same VIPA
7. end for
8. \( I \leftarrow \) set of servers under attack
9. \( G \leftarrow \frac{2w_i - \left| \{ S \mid i \in I \cap A \} \right|}{|A|} \) \( \Rightarrow \) Use at least \( G \) servers to serve all legitimate VIPAs.
10. for all server \( i \) under attack do
11. \( Q \leftarrow \) set of VIPAs on \( i \)
12. if \( |Q| = 1 \) then
13. \( S \leftarrow S \setminus Q \) \( \Rightarrow \) This VIPA is malicious
14. \( S \leftarrow S \setminus Q \) \( \Rightarrow \) This VIPA is malicious
15. Remove the malicious users
16. end if
17. \( V \leftarrow \) set of VIPAs on \( A \setminus I \); \( S \leftarrow S \setminus V \)
18. \( A \leftarrow \{ \text{server} 1, \ldots, \text{server} (K - G) \} \)
19. end while.

The basic idea of the Sequential Identification with Packing (SIP) algorithm can be sketched as follows. All the assignments of VIPAs are based on the adjustment of forwarding table at DAAs. Given unidentified VIPAs in set \( S \), first randomly assign them to available testing servers in set \( A \), with equal number of VIPAs on each server and no same VIPA on different servers. Hence each server gets roughly \( w_i = \left\lfloor \frac{|S|}{|A|} \right\rfloor \) VIPAs, and yields to a set of servers under attack \( I \). Then the algorithm gathers all VIPAs on safe servers
servers are safe.

The number of available testing servers $|\mathcal{G}|$ is non-decreasing. At the end of the algorithm, $|\mathcal{G}|$ remains unidentified. Since at the end of testing rounds, no other $K - |\mathcal{G}|$ legitimate VIPAs are tested out at each testing round. At most $K$ safe servers can be tested out at each testing round.

Algorithm Description.

The basic idea of the Sequential Identification without Packing (SIoP) algorithm can be sketched below. Given an initial set $S$ with $n$ users, we randomly assign them into $K$ server machines, with equal number of VIPAs on each server and no same VIPA on different servers. Conduct group tests on all VIPAs in $S$ and update $S$ to the unidentified VIPAs on servers under attack. Afterwards, SIoP reassigns all unidentified VIPAs in $S$ to $K$ servers, and leaves all legitimate VIPAs where they were since the time they were identified. Since $K > d$, at least $K - d$ safe servers can be tested out at each testing round and they may also contain legitimate VIPAs previously identified. Denote the set of VIPAs in $S$ as $V$. Since all VIPAs in $V$ are legitimate, if they are also in $S$, the algorithm then removes them from $S$. If any server under attack contains only one VIPA in $S$ (and many VIPAs in $V$), this VIPA in $S$ is malicious. The SIoP algorithm will get rid of this malicious user and remove it from $S$. This progress is repeated until all VIPAs in $S$ are identified. The pseudocode of SIoP is shown in Algorithm 2.

**3.2. Sequential Identification without Packing**

Since in the SIP algorithm, we move all legitimate users to good servers, it leaves exactly one malicious user (and none legitimate user) in an attacked server, thereby increasing the testing rounds. In this section, we still consider a networking scenario where each VIPA can be assigned to exactly one server. However, we also consider the server feasibility. As long as we guarantee that all servers (maybe under attack) serving these identified legitimate VIPAs are not overloaded, then we will leave these users there (rather than packing them into $G$).

**Lemma 3.1.** The number of available testing servers $|\mathcal{G}|$ is non-decreasing. Hence, the number of servers used for serving identified legitimate VIPAs, $|\mathcal{G}| = n - (|S| - (|A| - d)w_i)$ is non-decreasing. At the end of the algorithm, all legitimate VIPAs are identified, so $|\mathcal{G}|$ converges to $n - d$.

**Theorem 3.1.** The SIP algorithm can identify all malicious users within at most $O(\log \frac{n}{d})$ detection time, where $K' = K - \lceil \frac{n - d}{w} \rceil$.

**Proof.** At each round, at most $d$ servers are under attack, i.e., at most $\frac{d}{|S|}$ servers remain unidentified. Since at the end of the algorithm, all legitimate VIPAs are identified, hence at most $d$ malicious VIPAs remain suspect. If they are on different servers under attack, the machine with the largest weight will get rid of this malicious user and remove it from $S$. This progress is repeated until all VIPAs in $S$ are identified. The pseudocode of SIoP is shown in Algorithm 2.

**Remarks:** Compared with SIP, if no overloading happens, the SIoP algorithm avoids moving identified legitimate VIPAs and decreases the reassignment expense. However, if $w$ is relatively small, the algorithm also has to reassign all legitimate VIPAs to $K$ servers and balance the weight $w_i$ on different servers. Once overloading happens, the reason why reassigning legitimate VIPAs and suspect ones both into $K$ servers at two different testing rounds, instead of reassigning them together (all VIPAs) at one time, is to avoid the possibility that all safe servers after the reassignment contain no suspect VIPAs and much more legitimate VIPAs 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.

**Performance Analysis.**

Some proofs are omitted due to space limitation.

**Lemma 3.2.** The number of VIPAs $w_i$ on server $i$ does not exceed server capacity $w$ until round $j$ where $j = \log \frac{n}{w} = \frac{n}{n - w(K - d)}$.
3.3. Partial Non-adaptive Identification

In the previous two algorithms, we consider the scenario where each VIPA is assigned to exactly only one server. Therefore, during each testing round, we try to have exactly one malicious user (and many legitimate users) on each server. In this section, we consider a scenario where each VIPA can be assigned to multiple servers and allowed testing to be run in parallel, which is quite aggressive. In this case, each column in matrix $M$ can have many 1-entry, that is, $\sum_{i=1}^{K} M[i,j] \geq 1 \forall j$.

Consider the matrix $M_{t \times n}$ as discussed in Section 2.2. If $M$ is a $d$-disjunct matrix, then we can easily detect all malicious users in only one testing round where all pools are tested simultaneously, thus the detection time is $O(1)$. $M$ is called $d$-disjunct if no column is contained in the boolean sum of any other $d$ columns.

Algorithm 3 Construct a $d$-disjunct matrix

1: function dDisjunct(n)
2: Consider a finite field $GF(q)$, choose $s, q, k$ satisfying:
3: $kd \leq s \leq q$ and $n \leq q^k$
4: $\triangleright \text{Construct matrix } A_{t \times n}$
5: for $x \in \{0, s-1\}$ do
6: for all polynomials $p_j$ of degree $k$ do
7: $A[x, p_j] = p_j(x)$
8: end for
9: end for
10: $\triangleright \text{Construct matrix } M_{t \times n}$
11: for $x \in \{0, s-1\}$ do
12: for $y \in \{0, s-1\}$ do
13: for all polynomial $p_j$ of degree $k$ do
14: if $A[x, p_j] == y$ then
15: $M[(x, y), p_j] = 1$
16: else
17: $M[(x, y), p_j] = 0$
18: end if
19: end for
20: end for
21: end for
22: end function

Constructing a $d$-disjunct matrix has been studied extensively [4]. There is a relationship between the number of rows and the number of columns in $M$ as follows:

Lemma 3.5. [4] Given $d$ and $n$, the number of rows in a $d$-disjunct matrix with $n$ columns is lower bounded by:

$$t \geq \min \left\{ \left(\frac{d+2}{2}, n\right) \right\}$$

Therefore, in order to construct a $d$-disjunct matrix using $K$ servers and $n$ VIPAs, we will consider two cases: (1) $K \geq t$ and (2) $K < t$.

Case 1: $K \geq t$

When $K \geq t$, after the servers being attacked, we can use at most $t$ servers to construct a $d$-disjunct matrix $M_{t \times n}$. The assignment of $n$ VIPAs on $K$ servers will be based on $M_{t \times n}$. Thus the detection time is $O(1)$. For constructing a $d$-disjunct matrix, we resort to a classic algorithm using the
finite field. For the convenience of readers, we present this algorithm, called \textit{dDisjunct}, in Algorithm 3. We will use this algorithm as a sub-function when \( K < t \). Note that in this scheme, we assume that the row weight of constructed matrix does not exceed the server capacity \( w \).

**Lemma 3.6.** [4] Let \( t \) be the number of rows in the obtained \( d \)-disjunct matrix by Algorithm 3, then \( t = qs = O(q^2) \) where:

\[
q \leq (2 + o(1)) \frac{d \log_2 n}{\log_2(d \log_2 n)}
\]

**Case 2:** \( K < t \)

In this section, we present our Partially Non-adaptive Identification (PNI) algorithm. The basic idea of PNI can be sketched as follows. First equally assign \( n \) VIPAs to \( K \) servers, each VIPA is assigned to only one server. Assume we get a suspect VIPA set \( S \) at this round, pack all identified \( n - |S| \) legitimate VIPAs into \( G \) servers, hence \( G = \left\lfloor \frac{n - |S|}{w} \right\rfloor \). Apparently, these VIPAs need no test, so each of them is activated on only one server in \( G \).

Afterwards, if \( |A| = K - G \) servers are available for testing. If \( |A| \geq qs \) where \( q \leq (2 + o(1)) \frac{d \log_2 |S|}{\log_2(d \log_2 |S|)} \), \( kd \leq s \leq q \) and \( |S| \leq q^k \), we can construct a \( d \)-disjunct matrix \( M_{|A| \times |S|} \) by calling function \textit{dDisjunct}(\( |S| \)) in Algorithm 3 and identify all VIPAs in \( O(1) \) detection time. This is directly a non-adaptive testing method.

However, if \( A \) does not satisfy the above constraint, the PNI algorithm will perform the following steps: Find the maximum value \( n' \) \((1 \leq n' \leq |S|) \) such that \( |A| \geq qs \) where \( q \leq (2 + o(1)) \frac{d \log_2 |S|}{\log_2(d \log_2 |S|)} \), \( kd \leq s \leq q \) and \( n' \leq q^k \). Partition \( S \) into two disjoint sets \( S_1 \) and \( S_2 \) such that \( |S_1| = n' \). Then pack all \( S_2 \) to \( G \). Call function \textit{dDisjunct}(\( |S_1| \)) to identify all VIPAs in \( S_1 \) in \( O(1) \) detection time at this testing round. Pack identified legitimate VIPAs into \( G \) and use updated \( A \) available testing servers to test \( S = S_2 \). Iterate this process until all VIPAs are identified, thus detecting all malicious users.

**Performance Analysis.**

In this section, we will discuss the detection time complexity for scenarios with different \( K \) values. Our analysis are based on some assumptions and notations:

- assume \( n \equiv 0(\text{mod} \; K) \) and \( |A|w \equiv 0(\text{mod} \; d + 1) \) for convenience;
- assume \( n > w \geq d \geq 1 \) and \( |A| \geq d + 1 \);
- \( X \Leftrightarrow Y \) denotes that \( Y \) is the sufficient condition of \( X \).
- \( |A| \) is short for \( |A| \) at current testing round \( i \) in the following paragraphs.

**Lemma 3.7.** The number \( G \) of servers used for serving identified legitimate VIPAs is bounded by:\n
\[
G \leq \left\lfloor \frac{n-d}{w} \right\rfloor
\]

**Algorithm 4** Partial Non-adaptive Identification with \( K < t \)

1. Randomly assign \( n \) VIPAs to \( K \) servers, no two servers has the same VIPAs.
2. Pack all legitimate VIPAs on safe servers into \( G \) servers.
3. In \( G(q) \), choose \( q, s, k \) satisfying:
   
   \[
   q \leq (2 + o(1)) \frac{d \log_2 n}{\log_2(d \log_2 n)} \quad \text{and} \quad |S| \leq q^k
   \]
4. if \( |A| \geq qs \) then
   
   6. Run \textit{dDisjunct}(\( |S| \)), decode from the obtained \( d \)-disjunct matrix and identify all VIPAs; \( S \leftarrow 0 \)
   
    8. else
   
   9. Partition \( S \) into \( S_1, S_2 \) where \( |S_1| = n' \) and \( n' = \max \; n'' \) satisfying:
      
      \[
      |A| \geq qs, q \leq (2 + o(1)) \frac{d \log_2 n''}{\log_2(d \log_2 n'')}, \quad n'' \leq q^k
      \]
   
   10. Run \textit{dDisjunct}(\( |S_1| \)), decode from the obtained \( d \)-disjunct matrix and identify all VIPAs in \( S_1 \)
   
   11. Remove all malicious users in \( S_1 \). Pack legitimate VIPAs in \( S_1 \) to \( G \), update \( G \) and \( A \); \( S \leftarrow S_2 \)
   
   13. end while
   
   14. end if

**Lemma 3.8.** The number of available testing servers \( A \) is bounded by:

\[
K - \left\lfloor \frac{n-d}{w} \right\rfloor \leq |A| \leq K - \left\lfloor \frac{K-d}{w} \right\rfloor
\]

**Lemma 3.9.** (D’yachkov-Rykov lower bound) [11] Let \( n > w \geq d \geq 2 \) and \( t > 1 \) be integers. For any superimposed \((d - 1, n, w)\) code \((d, n, w)\) design \( X \) of length \( t \) \((X \) is called a \((d - 1)\)-disjunct matrix with \( t \) rows and \( n \) columns), the following inequality holds: \( t \geq \left\lfloor \frac{d^l}{w} \right\rfloor \)

**Corollary 3.1.** In the PNI algorithm, given \( w \geq d \geq 1 \), we have:

\[
n' = \min \left\{ \frac{|A|w}{d+1}, n - Kw + w + |A|w \right\}
\]

**Proof.** According to Lemma 3.9, when number of columns \( n'' \in [1, |S|] \), we have \( |A| \geq \left\lfloor \frac{(d+1)n''}{w} \right\rfloor \geq \frac{(d+1)n''}{w} \), so \( n' = \max \; n'' \leq \frac{|A|w}{d+1} \).

In the PNI algorithm, for each round \( i \), we have: \( |A_i| \geq K - \left\lfloor \frac{n-n'}{w} \right\rfloor \), thus \( n' \leq n - Kw + w + |A|w \). Therefore, if we can reach this upper bound, we have \( n' = \min \left\{ \frac{|A|w}{d+1}, n - Kw + w + |A|w \right\} \)

**Lemma 3.10.** At any testing round \( i \):

\[
(1) n'' = n - Kw + w + |A|w \quad \text{when} \quad K \in (d, \frac{dw+n+w}{d+1})
\]

\[
(2) |A|w \geq n - Kw + w + |A|w \quad \text{with} \quad K \in [k_1, \frac{dw+n+w}{d+1}]
\]

**Proof.** According to Corollary 3.1, \( n' = \min \left\{ \frac{|A|w}{d+1}, n - Kw + w + |A|w \right\}, \) then

\[
(1) K \leq \frac{dw+n+w}{w} \Rightarrow Kw \leq dw + n + w, \quad \text{and} \quad |A| \geq d + 1 \quad \Rightarrow \frac{|A|wd}{d+1} \geq wd, \text{hence}
\]

\[
Kw - n - w \leq \frac{|A|wd}{d+1}, \text{ so}
\]

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

\[
(2) \frac{|A|w}{d+1} \geq n - Kw + w + |A|w
\]

\[
\Leftrightarrow \frac{|A|}{K} \leq \frac{(K-1)(m-n)(d+1)}{Kw} \leq K - (K-d)n \leq Kw^2 - (dw + n + w)K - nd^2 \geq 0
\]
Solving this inequality, we get \( K \geq \frac{dn}{Kw} \), we need not do partitions in PNI algorithm. Since \( k_1 \leq \frac{dn}{Kw} + w \), so \( K \in [k_1, \frac{dn}{Kw} + w) \) holds for \( n' = n - K w + w + |A|w \).

Moreover, since \( k_1 > \frac{dn + w + w'}{w} \geq d^2 + w + w' \), there are no overlaps between these two intervals, thus proof completes.

Therefore, we split \( K \in (d, +\infty) \) into 4 divisions:

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

Next we will decide which subintervals of \( K \) yield \( O(1) \) detection time besides \( K \in [\frac{dn + w + w'}{w}, +\infty) \) above, and also time complexity for other \( K \) intervals.

**Lemma 3.11.** Besides \( K \in [\frac{dn + w + w'}{w}, +\infty) \), the PNI algorithm needs at most \( O(1) \) detection time with \( K \in [k_2, \frac{dn + w + w'}{w}] \), where \( d \leq \frac{w + \sqrt{w^2 - 4w(n - w)}}{2(n - w)} \) and \( k_2 = \frac{w + \sqrt{w^2 - 4w(n - w)}}{2w} \).

**Proof.** Since \( n' \geq \frac{dn}{Kw} \geq |S_0| \), all users are identified without partition in the PNI algorithm. Meanwhile, at least one server indicates being attacked at the first round, so \( |A_0| \geq K - \lfloor \frac{(K-1)n}{Kw} \rfloor \). With simple algebraic computations, we can reach the subinterval \( \left[k_2, \frac{dn + w + w'}{w}\right] \) on the condition that \( d \leq \frac{w + \sqrt{w^2 - 4w(n - w)}}{2(n - w)} \), within division I where \( n' = \lfloor \frac{|A|w}{d+1} \rfloor \); however, for division II and III, no detailed subintervals of \( K \) for \( O(1) \) detection time can be obtained.

**Lemma 3.12.** Within division I, PNI algorithm can identify all VIPAs with \( O(d + \frac{K}{\sqrt{n}}) \) detection time.

**Proof.** We derive the time complexity from the following recurrence:

**Starting round 0**: \( |S_0| \leq \frac{dn}{Kw} \), and \( K - \lfloor \frac{(K-1)n}{Kw} \rfloor \leq |A_0| \leq K - \lfloor \frac{(K-d)n}{Kw} \rfloor \)

**Ending round T**: \( 0 < |S_T| \leq \frac{|A_T|w}{d+1} \)

**Iteration**: For \( \forall i \in [0, T-1] \), \( |S_{i+1}| = |S_i| - \lfloor \frac{|A_i|w}{d+1} \rfloor \) and \( K - \lfloor \frac{n-|S_{i+1}|}{w} \rfloor \leq |A_{i+1}| \leq K - \lfloor \frac{n-|S_{i+1}| - d}{w} \rfloor \), hence

\[
\begin{align*}
|A_{i+1}| &\geq \frac{|S_i|}{w} - \frac{|A_i|}{d+1} - 1 \\
S_0 &\leq \sum_{i=0}^{T} \frac{|A_i|w}{d+1}
\end{align*}
\]

In order to estimate the maximum time cost, use \( |S_0| = \frac{dn}{Kw} \) to initiate the worst starting case. Solving this recurrence, we get the following inequality:

\[
KwT^2 - \left( \frac{dn}{Kw} + \frac{dn}{Kw} - K + \frac{n + w}{w} \right) T - \left( K - \frac{(K-1)n}{Kw} - \frac{dn(d+2)}{K} + w - 1 \right) \leq 0,
\]

where \( \alpha = \frac{Kw}{2(d+1)} + \frac{dn}{Kw} - K + w + \beta = 2Kw - 2K^2n + 2Kn - K w + 2Kw^2 - 2d(d + 2)nw \). Since \( \frac{n}{w} \leq K \), instead.

Therefore, PNI will complete the identification within at most \( O(d + \frac{K}{\sqrt{n}}) \) detection time.

**Remarks:** Since \( K \) is always much smaller than \( n \), the complexity will approach \( O(d) \) instead.

**Lemma 3.13.** Within division III, PNI can identify all VIPAs with at most \( O(d) \) detection time.

**Proof.** Similarly we derive the time complexity from the following recurrence:

**Starting round 0**: \( |S_0| = \frac{dn}{Kw} \), \( K - \lfloor \frac{(K-1)n}{Kw} \rfloor \leq |A_0| \leq K - \lfloor \frac{(K-d)n}{Kw} \rfloor \)

**Ending round T**: \( 0 < |S_T| \leq n - K w + w + |A_T|w \)

**Iteration**: For \( \forall i \in [0, T-1] \), \( |S_{i+1}| = |S_i| - (n - K w + w + |A_i|w) \) and \( K - \lfloor \frac{n-|S_{i+1}|}{w} \rfloor \leq |A_{i+1}| \leq K - \lfloor \frac{n-|S_{i+1}| - d}{w} \rfloor \), hence

\[
\begin{align*}
|A_{i+1}| &\geq \frac{|S_i|}{w} - |A_i| - 2 \\
S_0 &\leq \left(T + 1\right)(n - K w + w) + \sum_{i=0}^{T} |A_i|w
\end{align*}
\]

Solving this recurrence, we get the following inequality:

\[
\frac{2}{n + w} \left( \alpha' + \sqrt{-\beta'(n + w) + \sigma'^2} \right)
\]
Therefore, PNI will complete the identification within at most $O(d)$ detection time.

Corollary 3.2. Within division II, PNI algorithm can identify all VIPAs with $O(d + \frac{K}{\sqrt{w}})$ detection time.

Proof. According to Lemma 3.12 and 3.13, the time complexity of PNI algorithm depends on the value of $n'$ at each round, since $n'$ within division II oscillates between $\frac{|A|w}{d + 1}$ and $n - Kw + w + |A|w$, the time complexity is at most $O(d + \frac{K}{\sqrt{w}})$.

Theorem 3.3. Given $1 \leq d \leq w$, the PNI algorithm can identify all VIPAs within $O(d + \frac{K}{\sqrt{w}})$ detection time.

1) at most $O(d + \frac{K}{\sqrt{w}})$ detection time when $K \in (d, k_1)$, whilst at most $O(1)$ detection time when $K \in [k_2, \frac{dn + n + w}{w}]$ on condition that $d \leq \frac{w + \sqrt{w^2 - 4n^2(n-w)}}{2(n-w)}$;
2) at most $O(d)$ detection time when $K \in [k_1, \frac{dn + n + w}{w})$;
3) at most $O(1)$ detection time when $K \in [\frac{dn + n + w}{w}, +\infty)$. where

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

and

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

4. Discussions

Note that in practical, $\frac{K}{\sqrt{w}}$ is limited to small constant, so the PNI algorithm can identity all VIPAs within at most $O(d)$ detection time in some scenarios while $O(1)$ in others. Since $d$ is bounded, non-adaptive group testing greatly outperforms sequential group testing. Besides the assumption $w \geq d$ adopted in this paper, constructions of $d$-disjunct matrix under more universal conditions must be investigated. Meanwhile the study of false positive based on the Hamming distance will be included in our future work.

References