Security Mechanisms for Distributed Computing Systems

TOHOKU UNIVERSITY
Graduate School of Information Sciences

Security Mechanisms for Distributed Computing Systems

(分散計算システムのためのセキュリティ機構に関する研究)

A dissertation submitted for the degree of
Doctor of Philosophy (Information Sciences)

Department of Computer and Mathematical Sciences

by

Ling XU

January 16, 2012

Acknowledgments

I would like to sincerely thank Professor Hiroaki Kobayashi for giving me the opportunity to study at Tohoku University, persuading me to finish my PhD, instructing my research, accompanying me to international conferences, introducing me to internships and job opportunities, and patiently tolerating all my caprice and faults.

My deepest thanks must also go to Professor Hideaki Sone and Professor Takuo Suganuma for their instructions on my dissertation.

Associate Professor Hiroyuki Takizawa has guided me along my journey to Tohoku University. He spent days helping me arrange my accommodation, drove me to my entrance examination, provided me with sagacious advice each time I encountered problems in my research, and consoled me when I felt confused about my future. For these I am very grateful.

I would like to thank assistant Professor Ryusuke Egawa sincerely for solving my apartment crisis, spending precious time in proof-reading my papers and kindly providing me valuable advice on my life direction.

I am indebted to Dr. Yun Yang who accommodated me in Osaka University during the Great East Japan Earthquake and warmly encouraged me to pursue my dreams.

I would like to thank Dr. Yoshitomo Murata for accompanying me to international conferences, discussing with me on my master thesis and doctoral dissertation when I encountered problems, and kindly helping me to prepare my graduation oral defense.

Ms. Takako Shibayama at the Industry-University-Government Collaboration Division, Office of Cooperative Research and Development, Tohoku University has provided me with warm hearted help for which I am eternally grateful. But above all I thank her for treating me like her son.

I would like to thank Professors Toshikatsu Asai, Tomio Takahashi and Yukio Watanabe at the Innovative Leaders Platform in Tohoku University for providing me precious instructions and help on my career and life.

Also, one thing that you do need during your research is friendship, and I would like to thank all of my friends. In particular I would like to thank Ye Gao, Hong Wang, Qin Chen, Chao Qu, Jiali Yao, Lei Zhong for sharing their friendship with me and endowing me with many memories that I will cherish.

I would also like to thank my peers Katsuto Sato, Masayuki Sato, Yoshiei Sato and my tutor Yusuke Funaya, for all their warm hearted help and friendship over the past six years.

Finally, I would like to express my heart-felt gratitude to my mother, as I would not have made it to this point without her continued vast support.

Security Mechanisms for Distributed Computing Systems

Ling Xu

Abstract

Distributed computing systems (DCSs), such as peer to peer (P2P) systems and volunteer computing systems, are playing important roles in industry and our daily life. However, DCSs are vulnerable to the false result attack and the Sybil attack. In a DCS, under the false result attack, malicious worker nodes deliberately send incorrect results of computing tasks to host nodes. Under the Sybil attack, malicious users control many Sybil nodes to interfere the system. To ensure the application and the development of DCSs, it is necessary to address these two attacks.

The existing solutions to the false result attack are either inefficient or impractical. Existing mechanisms for resisting the false result attack are based on two techniques: Replication and Quiz. The Replication-based solutions enable hosts to distinguish correct results from incorrect ones. For a host and its workers, the host dispatches each task to multiple workers. Having received the results, the host chooses a result as being correct using a majority vote. The efficiency of the false result resisting mechanisms is defined as the percentage of unique tasks computed by workers among all the tasks computed by workers. Replication-based solutions face the problem of being inefficient, because each task is repeatedly computed multiple times. On the other hand, the Quiz-based solutions enable hosts to distinguish malicious workers from honest workers. For each host and its workers, the host sends to each worker a task set. Each task set contains some special tasks termed quizzes. Having received the results from workers, the host can judge a worker to be malicious if any result to the quizzes returned by this worker is incorrect. To implement Quiz-based solutions, however, it is required that quizzes satisfy certain special properties. How to generate quizzes that satisfy these properties is still an open problem.

Meanwhile, existing solutions to the false result attack are also problematic. Today, Sybil detecting algorithms that are based on the social network model (SNM) are the representative Sybil resisting solutions. This dissertation denotes these algorithms the SSD algorithms. SNM

is a model that depicts the network topologies of DCSs. In a DCS, the edges that connect nodes of different types (honest nodes and Sybil nodes) are called the attack edges. SNM assumes that the number of attack edges in the system is small. SSD algorithms aim to enable each honest node to judge the types of other nodes. In SNM-based DCSs, since the number of attack edges is small, the attack edges form a bottleneck that weakens the communication between nodes of different types. Hence, it is easier for honest nodes to communicate with other honest nodes than with Sybil nodes. Utilizing this property, honest nodes can distinguish honest nodes from Sybil nodes. The performances of SSD algorithms are evaluated by two metrics: honest accept rate (har) and Sybil accept rate (sar). har represents the average probability that honest nodes accept each other, and sar represents the average probability that honest nodes accept Sybil nodes. Here, two nodes accepting (rejecting) each other means that these two nodes regard each other to be honest (Sybil). The existing SSD algorithms face the problem of being inaccurate – they have high sar and low har. The bottleneck formed by the attack edges cannot completely prohibit the communication between nodes of different types. Hence, nodes make many incorrect judgments.

The objective of this dissertation is to create more effective mechanisms to resist the false result attack and the Sybil attack. To this end, this dissertation aims to design false result attack resisting mechanisms that are both efficient and quiz-free, and to create accurate SSD algorithms to resist the Sybil attack.

Chapter 2 proposes Mutual Spot Checking (MSC), an algorithm that enables hosts to detect malicious workers. The key idea is to use normal tasks, instead of quizzes, to check the types of workers. In MSC, hosts dispatch checking tasks (normal tasks) to each worker to compute. Then, hosts increase the reliabilities of workers that return correct results to the checking tasks. Since honest workers return more correct results, the reliabilities of honest workers will be higher than those of malicious workers. This enables hosts to distinguish honest workers from malicious ones. In MSC, quizzes are replaced with normal tasks, and only the checking tasks are repeatedly computed. Therefore, MSC is more practical than the Quiz-based solutions, and more efficient than the Replication-based solutions.

The performance of MSC is evaluated from three aspects: reliability gap, efficiency, and convergence. Theoretical analysis and simulations reveal that the reliabilities of honest workers are averagely higher than those of malicious workers in reasonable DCSs: 1). all DCSs when malicious worker do not collude, and 2). DCSs with malicious workers less than honest workers

when malicious workers can collude. Additionally, the theoretical efficiency of MSC is near optimal. Finally, simulation results show that the reliability of each worker converges to a stable value within ten rounds, which means that hosts can quickly identify malicious workers before accepting many incorrect results. These evaluation results validate that MSC is an efficient and practical solution to the false result attack.

Chapters 3 and 4 aim to create accurate SSD algorithms to resist the Sybil attack. To increase the accuracy of SSD algorithms, it is necessary to further prohibit the communication between nodes of different types. For this aim, the basic idea is to detect and cut the attack edges. Hence, the core innovation of Chapters 3 and 4 is an attack edge detecting technique – to enable honest nodes to distinguish the attack edges in the system.

Specifically, Chapter 3 proposes an attack edge detecting-based SSD algorithm called SybilDetector for authorized DCSs – DCSs that contain trustful authorities. The basic idea of SybilDetector is as follows. In a DCS, the shortest paths between nodes of different types have to pass the attack edges. Hence, in SybilDetector, two nodes accept each other only if the shortest paths between these two nodes do not pass the attack edges. In this way, honest nodes can accept each other and reject Sybil nodes.

The core of SybilDetector is an attack edge detecting mechanism based on the shortest path edge betweenness (SPEB), called SPEB-AED. In SybilDetector, nodes need to judge whether or not a certain edge is an attack edge. SPEB-AED is designed to realize this aim. The SPEB is a kind of edge betweenness metrics – metrics that measure certain properties of edges. In a DCS, for each pair of nodes and each shortest path between these two nodes, a message is transmitted along this shortest path. Then, the SPEB of each edge is defined as the number of messages passing through this edge. Previous research revealed that the SPEB satisfies a detecting property – the SPEBs of attack edges are higher than those of non-attack edges. Therefore, the problem of detecting attack edges is equal to the problem of detecting the edges with high SPEBs. Specifically, SPEB-AED has two steps. First, each node computes the SPEBs of the edges. Each node then computes a detecting threshold and regards the edges with SPEBs higher than its detecting threshold as attack edges. In this way, each honest node can detect attack edges and thus distinguish Sybil nodes from honest ones.

To evaluate SybilDetector, its accuracy is compared with that of SybilLimit on real world and synthetic network topologies. Here, SybilLimit is an existing representative SSD algorithm. The har of SybilDetector is comparable with that of SybilLimit. However, the sar of SybilDetector is

at least 4x and 10x lower than the sar of SybilLimit in the real world network topology and the synthetic network topology, respectively. These results not only confirm that SybilDetector is an accurate SSD algorithm, but also clarify the potential of the attack edge detecting technique in resisting the Sybil attack. It is expected that more effective Sybil resisting algorithms can be created using the attack edge detecting technique in the future.

Chapter 4 designs an attack edge detecting algorithm called Random walk and SNM-based Clustering (RSC) for unauthorized DCSs – DCSs that do not contain trustful authorities. In order to create accurate SSD algorithms, it is crucial to detect the attack edges. In authorized DCSs, it is feasible to detect the attack edges by computing the SPEBs of edges. In unauthorized DCSs, however, computing the SPEB is impossible because of the interference of Sybil nodes. The goal of Chapter 4 is to create an algorithm that enables each honest node to identify the possible attack edges among its incident edges in unauthorized DCSs. This algorithm can then be used to create accurate SSD algorithms for unauthorized DCSs.

To create this attack edge detecting algorithm, Chapter 4 takes two steps. The first step is to choose an edge betweenness metric that satisfies two properties: 1). this metric satisfies the detecting property, and 2). this metric can be securely computed in unauthorized DCSs under malicious interference. Such an edge betweenness metric is called a detecting metric. The second step is to enable each node to compute the betweennesses of its edges securely in a distributed manner. After these two steps, since the attack edges have high betweennesses, each node can identify the attack edges among its incident edges.

Specifically, Chapter 4 first chooses the random walk edge betweenness (RWEB) as the detecting metric for unauthorized DCSs. Like the SPEB, the RWEB is an edge betweenness metric defined by previous research. In a DCS, each pair of nodes disseminates a message to each other, where the message is transmitted between these two nodes in a random walk manner. Then, the RWEB of an edge is the expected number of messages passing through this edge. As all the messages between nodes of different types have to pass the attack edges, this dissertation expects that the RWEB satisfies the detecting property. Moreover, the RWEB is computed based on the information of random walk messages in the system. It is harder for Sybil nodes to interfere the computing of the information of random walk messages than to interfere the computing of shortest path information. This makes it possible to compute RWEBs in unauthorized DCSs. Based on these two observations, this dissertation uses the RWEBs as the detecting metric.

Then, RSC is designed to enable each node to securely compute the RWEBs of its incident

edges under malicious interferences. Representative attacks are discussed, and resisting mechanisms to these attacks are designed. Having obtained the RWEB knowledge, each node can now probabilistically distinguish the attack edges from its incident edges.

The performance of RSC is evaluated by the gap between the attack edge betweenness (aeb) and the honest edge betweenness (heb). Here, aeb (heb) represents the average of the betweennesses of attack edges (non-attack edges) computed by RSC. The gap should be large, so that the attack edges can be clearly differentiated from the non-attack edges. Simulations on real world and synthetic networks reveal that aeb is notably higher than heb. This result shows that RSC is a feasible attack edge detecting algorithm for unauthorized DCSs.

Chapter 4 then provides an example showing how RSC can be used to create accurate SSD algorithms. That is, RSC is embedded into SOHL – an existing SSD algorithm, to reduce the sar of SOHL. The new SSD algorithm is called RSC-based Sybil Resisting (RSSR). In SOHL, two nodes accept each other if and only if their random walk messages reach each other. Because of the existence of the bottleneck formed by the attack edges, the probability that messages of honest nodes reach Sybil nodes is low. Accordingly, honest nodes can distinguish Sybil nodes from honest ones. In RSSR, honest nodes first detect the attack edges using RSC. Then, honest nodes prevent their random walk messages from traversing the attack edges. Accordingly, it is expected that RSSR achieves a lower sar compared to SOHL. Simulation results show that the sar of RSSR is 5x and 13x lower than the sar of SOHL in the real world network topologies and synthetic network topologies, respectively. These results confirm the potential of RSC in creating accurate SSD algorithms.

The algorithms designed in this dissertation can effectively address the false result attack and the Sybil attack. Hence, this dissertation makes a stable contribution in promoting the application and development of DCSs.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

With the development of network topology, distributed systems like Facebook[1] and Skype[2] have been widely used in industry and our daily life. Specifically, a distributed system organizes multiple nodes together via networks. Each node v is incident (i.e., connected) to one or multiple nodes according to certain connection protocols. The nodes (edges) incident to v are called the *incident nodes* (incident edges) of v. v can only communicate with its incident nodes, and v knows nothing about the rest of the system. Nodes communicate with each other to finish certain goals.

Distributed Computing Systems

Distributed computing systems (DCSs) is a category of distributed systems that emphasize the sharing of computing power among nodes. DCSs are highly useful, where representative DCSs include *grid computing systems*[3], *volunteer computing systems*[4, 5], *P2P systems*[6], *sensor network systems*[7] and *ad hoc*

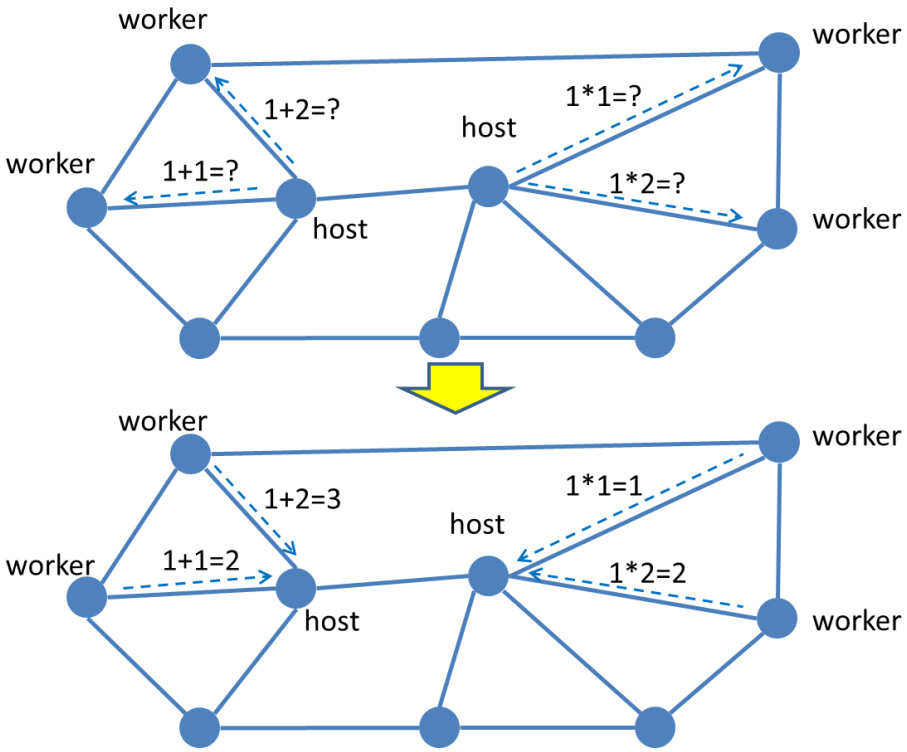


Figure 1.1: A volunteer computing system

systems[8].

For example, grid computing systems are DCSs that gather computing power of multiple administrative domains together to solve computing intensive problems. In industry and academia, many computing tasks, such as the development of new medicines, need significant computing power. Meanwhile, many computers in universities and companies are idling. Grid computing systems organize the computing power of these idling computers together to finish complicated computing tasks.

Similar to grid computing systems, volunteer computing systems organize idling computers on the Internet to finish computing tasks. Normally, nodes in grid computing systems belong to credible institutes such as universities. In contrast, nodes in volunteer computing systems are arbitrary computers on the

Internet. In a typical volunteer computing system such as BOINC[4], some nodes, denoted by *host* nodes, have computing tasks that need to be computed. Some nodes, denoted by *worker* nodes, have idling computing power. Hosts dispatch their tasks to workers to compute. In this way, idling computing power can be utilized. The most famous volunteer computing system today, SETI@home, is utilizing the computing cycles of millions of computers to search for extra-terrestrial intelligence[5]. Figure 1.1 shows an example of volunteer computing systems.

P2P systems denote the DCSs that are constructed in the *decentralized model*. Conventionally, DCSs are organized in the *centralized model*. In a centralized DCS, nodes are divided into two types: *servers* and *clients*. Servers provide resources to clients. Communication occurs only between servers and clients, and no communication occurs among clients. In contrast, in a pure P2P/distributed system, there is no server, and all nodes have equal privilege. Nodes communicate with each other and provide resources to each other to maintain the system. For example, P2P file-sharing systems like BitTorrent[9] enable nodes to share files with each other. In such a system, each file is located at a certain node. To fetch the file, a file-requesting node sends its request to a node that is "closer" to the file-holding node. The node receiving the request sends the request to another node that is even closer to the file holder. Finally, the request can reach the file holder.

To simplify the expression, this dissertation classifies DCSs into the following categories.

1. **Centralized DCSs**, **decentralized DCSs**, and **hybrid DCSs**: centralized DCSs and decentralized DCSs are DCSs that obey the centralized model and the decentralized model, respectively. In a hybrid model-based

DCS, most nodes have equal privilege and can communicate with each other. However, this system also contains servers that provide resources to other nodes. The online calling service Skype[2] is a representative hybrid DCS. In Skype, most nodes are equal, and they exchange messages with each other to ensure the quality of the communication. However, Skype also contains some "super nodes" that provide services to other nodes.

2. **Open DCSs** and **closed DCSs**: open DCSs do not set strict credibility criteria for member nodes. Any node of any organization can join these systems. Therefore, malicious nodes may exist in open DCSs and will launch attacks against the systems. Most volunteer computing systems and P2P systems are open DCSs. In contrast, closed DCSs only accept credible nodes. As introduced above, most grid computing systems belong to this category.

3. **Authorized DCSs** and **unauthorized DCSs**: this dissertation regards the servers in centralized DCSs and hybrid DCSs as trustful authorities, and calls DCSs that contain trustful authorities (no authority) the *authorized* (*unauthorized*) DCSs. Namely, centralized and hybrid DCSs are authorized DCSs, while decentralized DCSs are unauthorized DCSs. It is more difficult to resist attacks launched by malicious nodes in unauthorized DCSs than in authorized DCSs.

This dissertation concentrates on open DCSs, where security is a key problem. Especially, the *false result attack*[10, 11] and the *Sybil attack*[12] are two critical threats to open DCSs.

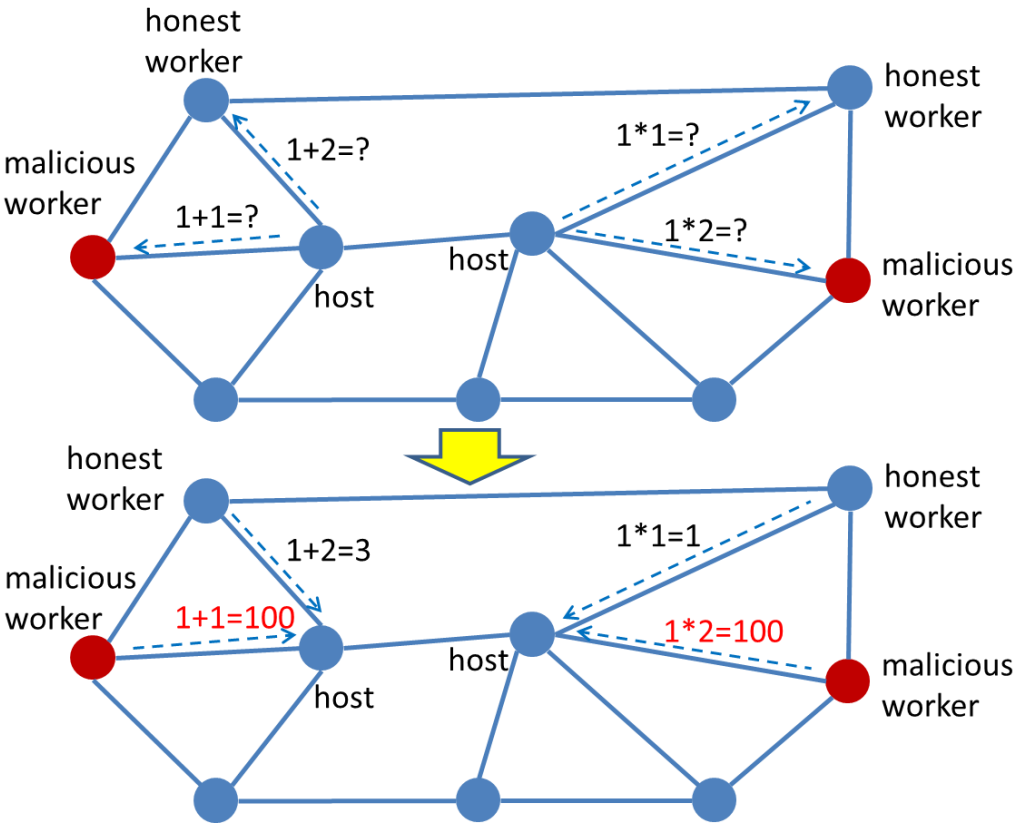


Figure 1.2: False result attack in volunteer computing systems

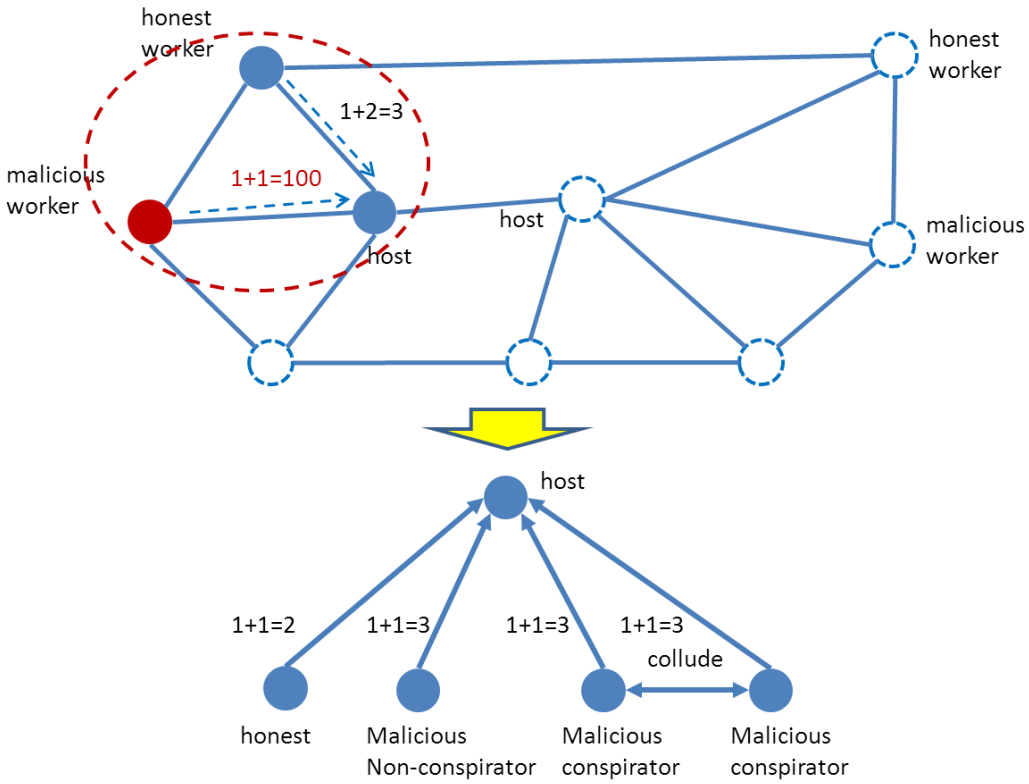


Figure 1.3: Model of the false result attack

False Result Attack

In the false result attack, malicious nodes deliberately disseminate incorrect data to honest nodes. For example, in a volunteer computing system, malicious workers can deliberately return incorrect results to hosts. Figure 1.2 shows an example of the false result attack in volunteer computing systems. In a file-sharing P2P system, malicious nodes can provide incorrect location information of files to the requester[13, 14].

Specifically, for a DCS, this dissertation discusses the false result attack in the following model. Each node v is considered as a host node, and the nodes that communicate with v are considered as the workers of v. Each host communicates with its workers by round. In each round, the host dispatches tasks to

workers. Workers compute their respective tasks and return the results to the host. Workers are divided into two types: *honest workers* and *malicious workers*. Honest workers always return correct results, while malicious workers may return incorrect results. Malicious workers are further divided into two types: *conspirators* and *non-conspirators*. Honest workers and non-conspirators can only communicate with the host, while conspirators can communicate with each other (collude). Figure 1.3 shows an example of this model. To simplify the expression, in the following discussion of the false result attack, this dissertation concentrates on the communication between a certain host v and its workers. Additionally, it is assumed that all nodes in the system are workers of v. Obviously, this assumption does not contradict the model of the false result attack. Henceforth, "the host" is used to denote node v, and "workers" is used to denoted the workers of v.

Existing false result attack resisting solutions are based on two core techniques: *Replication*[15] and *Quiz*[11]. The Replication-based solutions enable the host to distinguish correct results from incorrect ones. The host dispatches each task to multiple workers. Having received the results, the host chooses a result as the correct result using a majority vote. The efficiency of the false result resisting mechanisms is defined as the percentage of unique tasks computed among all the tasks computed. Replication-based solutions face the problem of being inefficient, because each task is repeatedly computed multiple times.

The Quiz-based solutions enable the host to distinguish malicious workers from honest ones. The host sends a task set to each worker. Each task set contains some special tasks termed *quizzes*. After receiving the results, the host can judge whether or not a worker w is malicious by checking whether the results of the quizzes in the task set of w are correct. Quiz-based solutions

are more efficient than Replication-based solutions. However, quizzes need to satisfy certain special properties, and how to generate quizzes is still an open problem[10, 11].

Sybil Attack

The Sybil attack is another security threat to DCSs, where a few malicious users control many malicious nodes to break the system protocols. In an open DCS, it is easy for a malicious user to create many malicious nodes. In the discussion of the Sybil attack, these malicious nodes are called *Sybil nodes*. Malicious users can control their Sybil nodes to defeat the system[16]. For example, when the number of Sybil nodes in the system is large, Sybil nodes can easily break the Replication-based false result attack resisting mechanisms. Figure 1.4 shows an example of the Sybil attack to the volunteer computing system.

Specifically, under the discussion of the Sybil attack, this dissertation considers a DCS that obeys the following *social network model* (SNM)[17]. Nodes in the system are divided into two types: honest nodes and Sybil nodes. Each honest node belongs to an honest user. There exists an undirected social network among honest nodes. An edge between two honest nodes reflects the trust relationship between these users in the real world. Call each edge that connects two nodes of different types an *attack edge*. The number of the attack edges is small. Hence, the attack edges largely separate the whole network into two regions: the *honest region* of honest nodes and the *Sybil region* of Sybil nodes. A DCS that obeys SNM is shown in Figure 1.5.

To resist the Sybil attack, SNM-based Sybil detecting (SSD) algorithms are drawing enormous attention from researchers[18, 19, 20]. For a DCS, SSD algorithms enable each honest node to judge the types of other nodes. In SNM-based

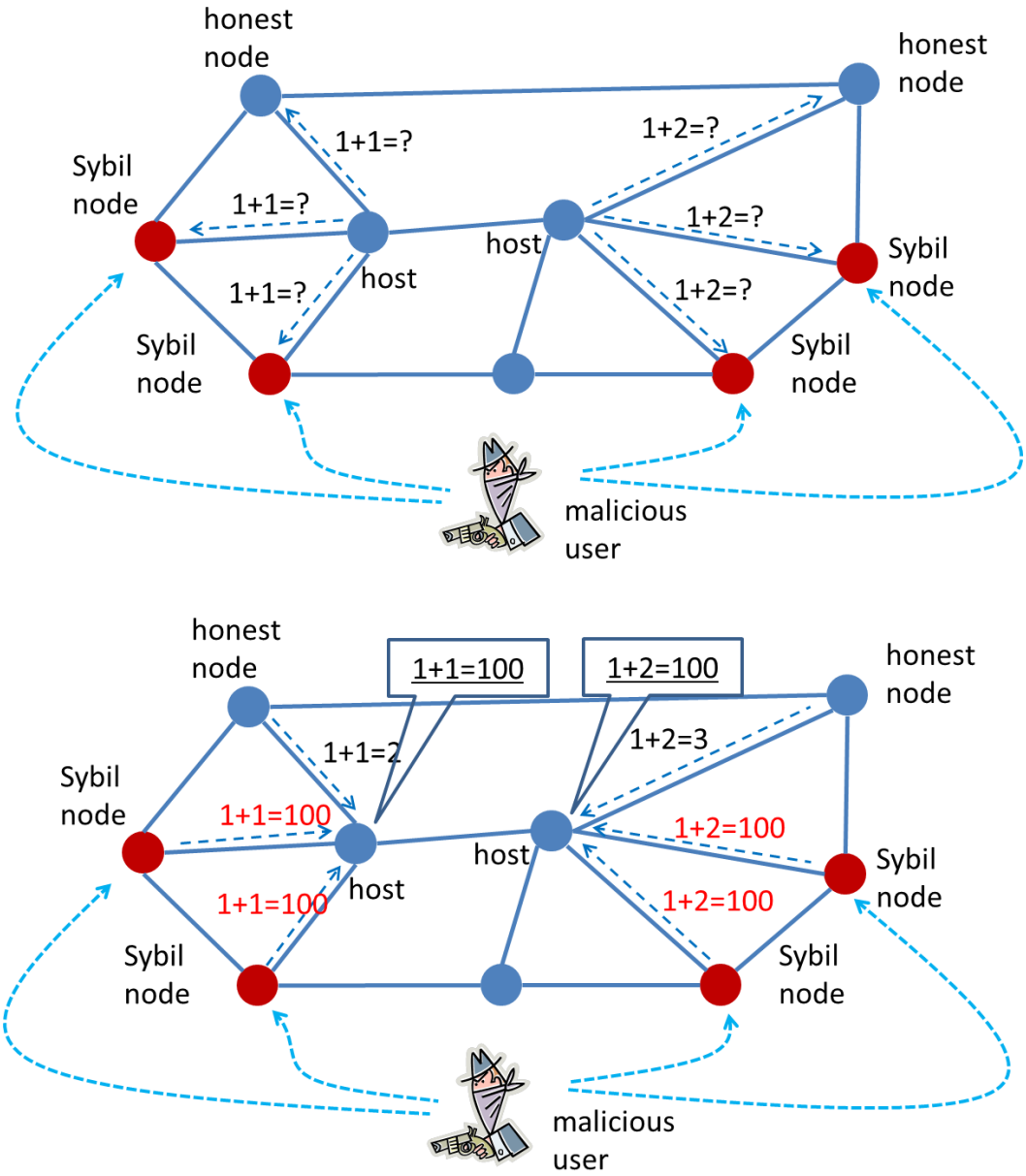


Figure 1.4: Sybil attack

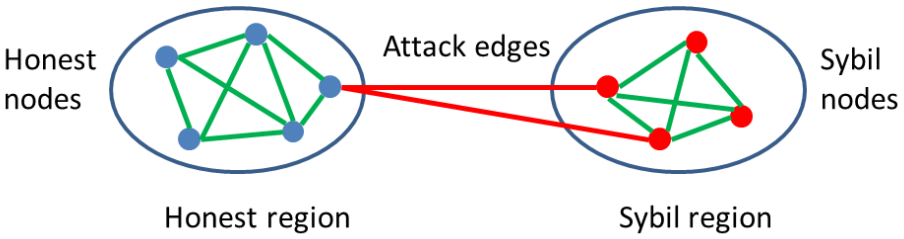


Figure 1.5: A distributed system obeying SNM

DCSs, since the number of attack edges is small, the communication between nodes of different types is weakened. Hence, it is easier for honest nodes to communicate with honest nodes than with Sybil nodes. Utilizing this property, honest nodes can distinguish honest nodes from Sybil nodes.

The performances of SSD algorithms is measured by their accuracy. For the simplicity of expression, for two random nodes v and u, it is said that v accepts (rejects) u if v regards u to be honest (Sybil). The performances of SSD algorithms are measured by the *honest accept rate* (har) and the *Sybil accept rate* (sar). har represents the average probability that v accepts u, where v and u are both honest. sar represents the probability that v accepts u, where v is honest and u is Sybil. SSD algorithms should have high har and low sar.

The problem of existing SSD algorithms is that their accuracy is low. Although the attack edge bottleneck can weaken the communication between nodes of different types, it cannot entirely stop the communication between nodes of different types. Accordingly, it is possible for honest nodes to make incorrect judgments.

To ensure the secure application and the development of DCSs, it is necessary to design more effective mechanisms to address the false result attack and the Sybil attack.

1.2 Objective of the Dissertation

The objective of this dissertation is to design more effective mechanisms to resist the false result attack and the Sybil attack on DCSs.

Specifically, to resist the false result attack, this dissertation aims to design a practical and efficient algorithm that enables the host to detect malicious workers. Here, "practical" means that no quizzes are used. In this algorithm, the host computes the *reliability* of each worker. The performance of this algorithm is mainly evaluated by the gap between the reliabilities of honest workers and malicious workers. The reliabilities of honest workers should be notably higher than those of malicious workers, enabling the host to detect malicious workers.

To resist the Sybil attack, this dissertation aims to create accurate SSD algorithms by utilizing an *attack edge detecting* technique. Here, for a DCS, attack edge detecting means to enable honest nodes to judge whether or not a certain group of edges are attack edges. This dissertation observes that attack edge detecting plays an important role in creating accurate SSD algorithms. Hence, a SSD algorithm should contain two components: an attack detecting mechanism and a distinguishing mechanism. For each honest node v, the attack edge detecting mechanism enables v to detect the attack edges in the system. Then, the distinguishing mechanism enables v to decide whether or not node u is Sybil.

Specifically, this dissertation discusses authorized DCSs and unauthorized DCSs separately. For authorized DCSs, an attack edge detecting-based SSD algorithm is to be created, which is expected to have high har and low sar. For unauthorized DCSs, an attack edge detecting algorithm is to be designed. This algorithm enables nodes to compute the *betweennesses* of edges. It is expected that the betweennesses of attack edges are notably higher than those of non-attack edges, which enables nodes to detect the attack edges. This attack edge

detecting algorithm can be used to create accurate SSD algorithms for unauthorized DCSs.

The mechanisms proposed above can more effectively address the false result attack and the Sybil attack. Accordingly, this dissertation can further promote the application and development of DCSs.

1.3 Organization of the Dissertation

This dissertation contains five chapters. Chapter 1 introduces the background knowledge and the objective of this dissertation. Chapter 2 proposes *Mutual Spot Checking* (MSC), a false result attack resisting algorithm that enables the host to distinguish malicious workers from honest ones. Chapter 3 and Chapter 4 deal with the Sybil attack. Chapter 3 designs SybilDetector, an attack edge detecting-based SSD algorithm for authorized DCSs. Chapter 4 designs *Random walk and SNM-based Clustering* (RSC), an attack edge detecting algorithm for unauthorized DCSs. Finally, Chapter 5 concludes this dissertation.

Chapter 2

MSC: a Practical Spot Checking Mechanism

2.1 Introduction

The false result attack is a key threat to DCSs, where malicious workers return incorrect results of tasks to the host deliberately. This attack is even harder to resist when conspirator workers cooperate with each other to break the system protocols.

To resist the false result attack, many existing solutions are based on the technique of Replication[15]. In existing Replication-based solutions, the host dispatches each task to $k = 2m + 1$ workers. After receiving the results, the host accepts a result that repeats more than m times. In this way, the host can filter out incorrect results. The main problem of Replication-based solutions is that they are inefficient, because each task has to be computed multiple times. Specifically, the efficiency of false result attack resisting algorithms is defined

as

$$\frac{\text{\# of unique tasks computed by workers}}{\text{\# of all tasks computed by workers}}.$$

Obviously, the optimal efficiency of false result attack resisting solutions is one.

Another core technique for false result attack resisting mechanisms is Quiz[11, 21, 22, 23]. In existing Quiz-based solutions, the host checks the correctness of results of the quizzes returned from workers to judge the types of workers. Accordingly, a quiz has to satisfy the *one-way complexity* property: the correctness of a quiz can be easily validated. Meanwhile, a quiz also has to satisfy the *non-distinguishableness* property: it is impossible for workers to distinguish whether or not a task is a quiz. However, how to generate tasks that satisfy these two properties is still an open problem.

This chapter aims to provide an efficient and quiz-free algorithm that enables the host to detect malicious workers. To this end, this chapter proposes Mutual Spot Checking (MSC). The key idea is to use normal tasks, instead of quizzes, to judge the types of workers. In MSC, the host dispatches *checking tasks* (normal tasks) to each worker to compute, and increases the reliabilities of workers that return correct results to the checking tasks. Additionally, workers, instead of the host, check the correctness of results of the checking tasks. Since honest workers return more correct results, the reliabilities of honest workers will be higher than those of malicious workers, enabling the host to distinguish honest workers from malicious ones. In MSC, quizzes are replaced by normal tasks, and only the checking tasks are computed multiple times. Therefore, MSC is more practical than the Quiz-based solutions, and more efficient than the Replication-based solutions.

The performance of MSC is measured from three aspects: efficiency, reliability gap, and the convergence performance. The efficiency of MSC should be high.

Additionally, the gap between the average reliability of honest workers and that of malicious workers should be large, so that the host can distinguish malicious worker accurately. Finally, the reliability of each worker should quickly converge to a stable value as the system runs. Hence, the host can detect malicious workers before accepting too many incorrect results.

The performance of MSC is evaluated by theoretical analysis and simulations. Collusion DCSs and non-collusion DCSs are analyzed, respectively. First, the evaluation shows that, in a non-collusion DCS, the host can detect all malicious workers. In a collusion DCS, the host can detect malicious workers as long as the number of malicious workers is less than that of honest workers. Additionally, the theoretical efficiency of MSC can approach one. Finally, simulation results reveal that reliabilities of workers can converge to stable values within ten rounds.

The organization of this chapter is as follows. Section 2.2 provides more discussion on existing false result attack resisting mechanisms. Section 2.3 explains the design of MSC. Section 2.4 and Section 2.5 analyze the performance of MSC theoretically and experimentally, respectively. Finally, Section 2.6 concludes this chapter.

2.2 Resisting Mechanisms to False Result Attack

Golle et al.[11] proposed a Quiz-based algorithm called *magic number* (MN) that enables the host to identify malicious workers. In MN, all tasks are assumed to satisfy the one-way complexity property. Therefore, the host can easily check the correctness of results returned from workers, and thus identify malicious workers. However, in general DCSs, tasks do not satisfy the one-way complexity property.

In the work of Zhao et al.[10], for each worker w, the host disseminates a set of tasks. Among this set of tasks, some tasks are quizzes. Having received the results from w, the host regards w to be malicious if any result to the quizzes is incorrect. However, this algorithm has two problems. First, the host itself has to check the correctness of the results of the quizzes. Second, the quizzes are assumed to satisfy the non-distinguishableness property, but how to generate tasks that satisfy this property is not yet clear.

Silaghi et al.[15] created a Replication-based malicious worker detecting algorithm, denoted by Λ. In Λ, the host dispatches each task to three random workers. If the result returned by w is equal to the results returned by the other two workers, it is said that w has made a match. Honest workers will make more matches than malicious workers. Accordingly, in a long run, malicious workers will have notably lower reliabilities and thus will be detected by the host. Λ is similar to MSC. However, since each task is computed three times, the efficiency of Λ is not satisfiable.

To resist the false result attack, beside the Quiz and Replication based solutions, another group of solutions aims to protect the task-solving application from being tampered by workers. Specifically, in DCSs, each worker has an application. On receiving a task, the worker runs the application using the task

as the input. The application generates the answer and sends it to the host. To make the application generate incorrect answers, malicious workers need to tamper with it. Therefore, the false result attack can be resisted by preventing workers from tampering with the application.

Both software and hardware protection mechanisms have been proposed. From the software perspective, *tamper proofing* and *obfuscation* are two well investigated techniques for code protection. The goal of the former technique is to make applications nonfunctional once the applications are modified. The goal of the latter technique is to encrypt applications without changing their functionalities. A general survey on the topics of the code protection can be found in [24]. However, so far, these software solutions are far from practical[25, 26, 11]. The hardware-based solutions aim to protect the applications using special hardware devices[27, 28]. An example is to let workers run applications using the *cell processor*[29], where the cell processor can protect the code running from being tampered by any outsize force. However, so far, hardware protection devices are not widely available.

Table 2.1: Networks for evaluation

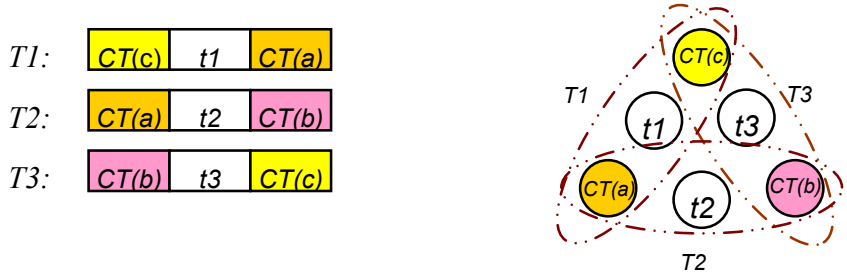
Name	Description
N_{ct}	The number of checking tasks shared by a pair of workers
N_{ts}	The number of tasks contained in each task set
N_p	The number of workers in the system
w	Each non-conspirator computes w percent of tasks in its task set
P_f	The percentage of malicious workers in the system
P_c	The percentage of conspirators among malicious workers
ϕ_i^n	The reliability of worker i in the n-th round
$\phi_H^n, \phi_M^n, \phi_C^n, \phi_{NC}^n$	The expected reliabilities of honest workers, all malicious workers, conspirators and non-conspirators in the n-th round, respectively

2.3 Mutual Spot Checking

This section explains the design of MSC. Table 2.1 lists the important parameters used in this chapter.

2.3.1 Basic Idea

The basic idea of MSC is as follows. In each round, the host dispatches to each worker a set of tasks, called a *task set*. Each pair of task sets dispatched to two workers share some tasks in common, called the checking tasks. An example of three task sets is shown in Figure 2.1. For each worker, the host maintains a *reliability value* for this worker. After receiving the results, for each pair of workers i and j, the host compares the results of the checking tasks shared by (the task sets of) i and j. It is said that "i matches the k-th checking task with j" if the results of the k-th checking task shared by i and j are identical. The host



(a) Three task sets to be dispatched to three workers.

(b) The relation between the three task sets.

Figure 2.1: Three task sets (T). Each task set contains three tasks (t). Each pair of tasks shares one checking task CT.

increases the reliabilities of i and j if they have made matches.

In a long run, the reliabilities of honest workers will be higher than those of malicious workers. The reason is intuitive. The results of the checking tasks shared by two honest workers must be identical. In contrast, the results of the checking tasks shared by two workers are inclined to be different, if any one of them is malicious. Accordingly, honest workers make more matches and have higher reliabilities. Given the reliability of each worker, the host can detect malicious workers.

MSC is a combination of the Replication and the Quiz methodologies. MSC is more efficient than the replication method, because only the checking tasks need to be executed by different workers. Additionally, it is more practical than the Quiz-based solutions, since quizzes are replaced by normal checking tasks.

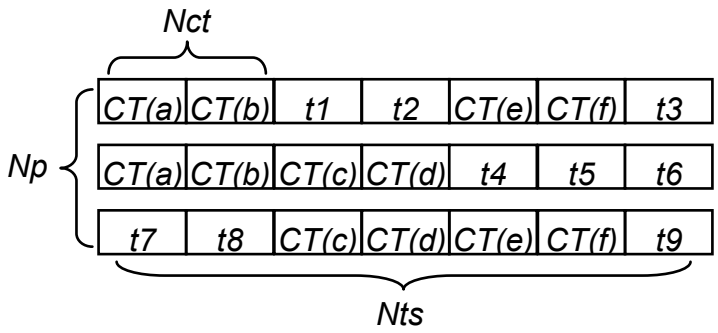


Figure 2.2: Instance of the checking task dispatching pattern with $N_p = 3$, $N_{ct} = 2$ and $N_{ts} = 7$.

2.3.2 The Way to Form Task Sets

One key issue with MSC is to find a way to dispatch the checking tasks so that high efficiency and detecting accuracy can be achieved. The dispatching patterns can be specified by three parameters: the number of workers in the system (N_p), the number of checking tasks shared by a pair of workers (N_{ct}), and the number of tasks contained in a task set (N_{ts}). Intuitively, the rate of N_{ts}/N_{ct} reflects the trade-off between the efficiency and detecting accuracy. When N_{ts}/N_{ct} is high, the efficiency of MSC is high because fewer tasks are repeatedly computed. Contrarily, when N_{ts}/N_{ct} is low, nodes compute more checking tasks. Accordingly, the host can detect malicious workers more quickly. There are quite a large number of possible combinations for these three parameters. As the first step to establish an effective assignment scheme, a typical and regular pattern is investigated in this chapter. Figure 2.2 presents an instance of such a regular pattern.

2.3.3 Reliability Metric

This section discusses how to compute the reliabilities of workers. Naturally, the reliability should satisfy the following properties:

1. A worker will obtain a higher reliability if it returns more matching results.

2. A worker who matches a checking task with an honest worker (worker with a higher reliability) obtains a higher reliability than those who match with a malicious worker.

Accordingly, the reliability of worker i in the n-th round, denoted by ϕ_i^n, is defined as

$$\phi_i^n = E(\frac{\sum_{0 \le j \ne i \le N_p} \sum_{1 \le i \le N_{ct}} c_i^k(j) \cdot \phi_j^{n-1}}{\sum_{0 \le j \ne i \le N_p} N_{ct} \cdot \phi_j^{n-1}}), \tag{2.1}$$

where $E(\cdot)$ means the probability expectation, and $c_i^k(j)$ is an indicator random variable defined by

$$c_i^k(j) = \begin{cases} 1: & P_i \text{ and } P_j \text{ match their } k\text{-th checking task. } k \in [1, N_{ct}] \\ 0: & \text{otherwise.} \end{cases}$$

Table 2.2: Three scenarios to study

Scenario 1	Non collusion	
Scenario 2	Collusion	$N_{ts} \cdot w - N_p \cdot P_f \cdot P_c \cdot N_{ct} \leq 0$
Scenario 3	Collusion	$N_{ts} \cdot w - N_p \cdot P_f \cdot P_c \cdot N_{ct} > 0$

2.4 Analysis

This section evaluates the theoretic performances of MSC in non-collusion and collusion DCSs. In a non-collusion DCS, workers are divided into two groups: honest and malicious. In a collusion DCS, malicious workers are further classified into conspirators and non-conspirators. Let P_f be the percentage of malicious workers in the system, and P_c be the percentage of conspirators among malicious workers.

In the collusion DCS, non-conspirator workers randomly pick exact w percentage of tasks in their task sets to compute, while conspirators compute *at least* w percent of tasks. Conspirators will first compute all the checking tasks known from their accomplices. If these tasks have accounted for more than w percent of the total N_{ts} tasks in their task sets, they stop computing; otherwise, they continue to pick the remaining tasks randomly from their task sets until w percent of tasks are accomplished. The former situation corresponds to the case where $N_{ts} \cdot w - N_p \cdot P_f \cdot P_c \cdot N_{ct} \leq 0$, and the latter satisfies $N_{ts} \cdot w - N_p \cdot P_f \cdot P_c \cdot N_{ct} > 0$. Therefore, according to a) whether the system is under collusion, and b) whether $N_{ts} \cdot w - N_p \cdot P_f \cdot P_c \cdot N_{ct} < 0$, three scenarios referred to as Scenarios 1, 2 and 3, shown in Table 2.2, are discussed.

2.4.1 Scenario 1 (Non-Collusion DCS)

This section proves that, in non-collusion DCSs, MSC can accurately distinguish all malicious workers.

Equation (2.1) can be rewritten as

$$\phi_i^n = E(\frac{\sum_{P_j \in H, j \neq i} \sum_{k=1}^{N_{ct}} c_i^k(j) \phi_j^{n-1} + \sum_{P_j \in M, j \neq i} \sum_{k=1}^{N_{ct}} c_i^k(j) \phi_j^{n-1}}{\sum_{P_j \in H, j \neq i} N_{ct} \cdot \phi_j^{n-1} + \sum_{P_j \in M, j \neq i} N_{ct} \cdot \phi_j^{n-1}}). \tag{2.3}$$

According to the types of workers i and j, $E(c_i^k(j))$ can be three values:

1. When both workers i and j are malicious, $E(c_i^k(j)) = 1 \cdot \Pr(c_i^k(j) = 1) + 0 \cdot \Pr(c_i^k(j) = 0) = w^2$, where Pr means the possibility;
2. When both workers i and j are honest, $E(c_i^k(j)) = 1$;
3. When one of workers i and j is malicious, supposing that worker i is honest (for j is the same), $E(c_i^k(j)) = w$.

Accordingly, Equation (2.3) can be written as

$$\phi_H^n = \frac{(1 - P_f) \cdot \phi_H^{n-1} + P_f \cdot w \cdot \phi_M^{n-1}}{(1 - P_f) \cdot \phi_H^{n-1} + P_f \cdot \phi_M^{n-1}}, \tag{2.4}$$

$$\phi_M^n = \frac{(1 - P_f) \cdot w \cdot \phi_H^{n-1} + P_f \cdot w \cdot w \cdot \phi_M^{n-1}}{(1 - P_f) \cdot \phi_H^{n-1} + P_f \cdot \phi_M^{n-1}} = w \cdot \phi_H^n, \tag{2.5}$$

where ϕ_H^n and ϕ_M^n denote the expected reliabilities of honest and malicious workers in the n-th round, respectively.

The solution of Equations (2.4) and (2.5) is

$$\phi_H = \frac{P_f \cdot w^2 - P_f + 1}{P_f \cdot w - P_f + 1}, \tag{2.6}$$

$$\phi_M = w \cdot \phi_H. \tag{2.7}$$

Equations (2.6) and (2.7) reveal two things. First, the reliabilities of honest and malicious workers are independent of parameters N_p, N_{ct} and N_{ts}, but are solely determined by P_f and w. Second, the average of the expected reliabilities of malicious workers is always w times less than that of honest workers. This means that, statistically, MSC is always able to distinguish malicious workers in the non-collusion DCS.

2.4.2 Scenario 2 ($N_{ts} \cdot w - N_p \cdot P_f \cdot P_c \cdot N_{ct} \leq 0$)

This section shows that, in Scenario 2, the average reliability of honest workers is expected to be larger than the conspirators as long as malicious workers are less than honest workers.

Under the collusion DCS, let ϕ_H^n ,ϕ_C^n and ϕ_{NC}^n be the reliabilities of honest workers, malicious conspirators and malicious non-conspirators in the n-th round, respectively. Now Equation (2.1) changes to

$$\phi_H^n = \frac{(1-P_f)\cdot\phi_H^{n-1} + P_f\cdot(1-P_c)\cdot w\cdot\phi_{NC}^{n-1}}{(1-P_f)\cdot\phi_H^{n-1} + P_f\cdot P_c\cdot\phi_c^{n-1} + P_f\cdot(1-P_c)\cdot\phi_{NC}^{n-1}}, \tag{2.8}$$

$$\phi_C^n = \frac{P_f\cdot P_c}{(1-P_f)\cdot\phi_H^{n-1} + P_f\cdot P_c\cdot\phi_C^{n-1} + P_f\cdot(1-P_c)\cdot\phi_{NC}^{n-1}}\cdot\phi_C^{n-1}, \tag{2.9}$$

$$\phi_{NC}^n = w\cdot\phi_H^n, \tag{2.10}$$

which has two sets of solutions. When the coefficient of ϕ_H^{n-1} in Equation (2.8) is larger than that of ϕ_C^{n-1} in Equation (2.9), namely when

$$P_f \leq 1/(1 + P_c + (P_c - 1)\cdot w^2), \tag{2.11}$$

ϕ_C converges to zero more quickly than ϕ_H. Then, the solution of Equations 2.8, 2.9 and 2.10 is

$$\phi_H = \frac{-P_f \cdot w^2 + P_f \cdot P_c \cdot w^2 + P_f - 1}{-P_f \cdot w + P_f \cdot P_c \cdot w + P_f - 1}, \phi_C = 0, \phi_{NC} = w \cdot \phi_H; \tag{2.12}$$

otherwise, the solution is

$$\phi_H = 0, \phi_C = 1, \phi_{NC} = 0. \tag{2.13}$$

Note that $\frac{1}{1+P_c+(P_c-1)\cdot w^2} \geq 1/2$ because $P_c \in [0,1]$ and $w \in [0,1]$. By Equation (2.11), it is concluded that the reliability of an honest worker is expected to be larger than that of a conspirator as long as P_f is less than 0.5.

2.4.3 Scenario 3 ($N_{ts} \cdot w - N_p \cdot P_f \cdot P_c \cdot N_{ct} > 0$)

This section shows that, in Scenario 3, MSC can distinguish malicious workers as long as malicious workers are less than honest workers.

Equation (2.1) becomes

$$\phi_H^n = \frac{(1-P_f) \cdot \phi_H^{n-1} + P_f \cdot P_c \cdot w' \cdot \phi_C^{n-1} + P_f \cdot (1-P_c) \cdot w \cdot \phi_{NC}^{n-1}}{(1-P_f) \cdot \phi_H^{n-1} + P_f \cdot P_c \cdot \phi_C^{n-1} + P_f \cdot (1-P_c) \cdot \phi_{NC}^{n-1}}, \tag{2.14}$$

$$\phi_C^n = \frac{(1-P_f) \cdot w' \cdot \phi_H^{n-1} + P_f \cdot P_c \cdot \phi_C^{n-1} + P_f \cdot (1-P_c) \cdot w \cdot w' \cdot \phi_{NC}^{n-1}}{(1-P_f) \cdot \phi_H^{n-1} + P_f \cdot P_c \cdot \phi_C^{n-1} + P_f \cdot (1-P_c) \cdot \phi_{NC}^{n-1}}, \tag{2.15}$$

$$\phi_{NC} = w \cdot \phi_H, \tag{2.16}$$

where

$$w' = \frac{N_{ts} \cdot w - N_p \cdot P_f \cdot P_c \cdot N_{ct}}{N_{ts} - N_p \cdot P_f \cdot P_c \cdot N_{ct}}$$

is the percentage of tasks that a conspirator will continue to compute after it has finished the checking tasks known from its accomplices.

Equations (2.14) and (2.15) indicate that as P_f approaches zero, $\lim_{P_f \to 0} \phi_H = 1$, $\lim_{P_f \to 0} \phi_C = w'$, meaning that ϕ_H is larger than ϕ_C when P_f is closer to zero. On the other hand, $\phi_H^n = \phi_C^n$ only when $P_f = 1/(1 + P_c + (P_c - 1) \cdot w^2)$, which is larger than $1/2$. Accordingly, the reliabilities of honest workers are larger than those of malicious workers as long as P_f is less than 0.5.

In brief, in this section, the performance of MSC is theoretically analyzed, and two notable points are revealed. Call 1). non-colluding DCSs and 2). colluding DCSs where malicious workers are less than honest workers the *reasonable DCSs*. First, in reasonable DCSs, the host can accurately detect malicious workers. Second, in reasonable DCSs, theoretically, MSC can achieve an optimal efficiency. In these DCSs, the average reliability of honest workers is always higher than that of malicious workers, no matter how N_{ct} and N_{ts} change. Higher N_{ts}/N_{ct} means a higher efficiency. Therefore, by enlarging N_{ts}, it is theoretically possible to increase the efficiency of MSC to one.

2.5 Evaluation

This section evaluates the reliability gap between honest workers and malicious workers by simulation. This gap is expected to be large. The converging performance of MSC is also evaluated by simulation. It is expected that the reliability of each worker quickly converges to a stable value.

In all the evaluations, by Equation (2.1), the reliability of a worker i in the n-th round is computed by

$$\phi_i^n = \frac{\sum_{0 \leq j \neq i \leq N_p} \sum_{1 \leq i \leq N_{ct}} c_i^k(j) \cdot \phi_j^{n-1}}{\sum_{0 \leq j \neq i \leq N_p} N_{ct} \cdot \phi_j^{n-1}}. \tag{2.17}$$

The probability expectation of Equation (2.1) is interpreted as the mean of the reliabilities of this worker from the first round until the current round.

2.5.1 Reliability Gap

Scenario 1

Figure 2.3 indicates the changes in reliabilities of different types of workers as w increases in the case where MSC is used for Scenario 1. The system consists of 40 workers. Each task set contains 400 tasks, and each pair of task sets share eight checking tasks. The percentage of malicious workers P_f is set at 0.5. The results show that honest workers obtain higher reliabilities than malicious workers as long as w is less than 1, which matches the analysis well.

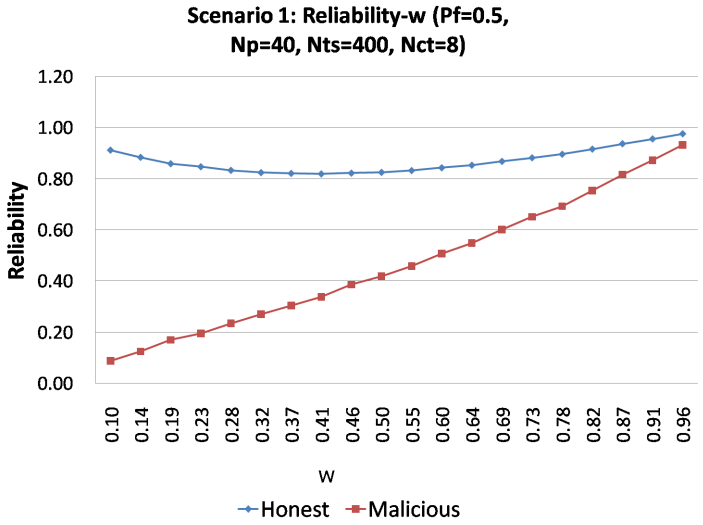


Figure 2.3: Changes in the reliabilities of honest and malicious workers with the increase of w in Scenario 1.

Scenario 2

Figure 2.4 is an example of running MSC in Scenarios 2. The system has 40 workers. The task set size is 320, and each pair of task sets has eight checking tasks in common. w is set at 0.2. Conspirator rate P_c is fixed at 0.5. This figure shows the changes in reliability of each worker type, as P_f increases. It reviews that P_f in the system is a critical factor affecting the feasibility of MSC.

In particular, when P_f is low (lower than 70 percent in this example), the average of the expected reliabilities of honest workers is much higher than that of malicious ones. Moreover, conspirators show no advantage over these non-conspirators. Therefore, the host can clearly identify malicious workers in this

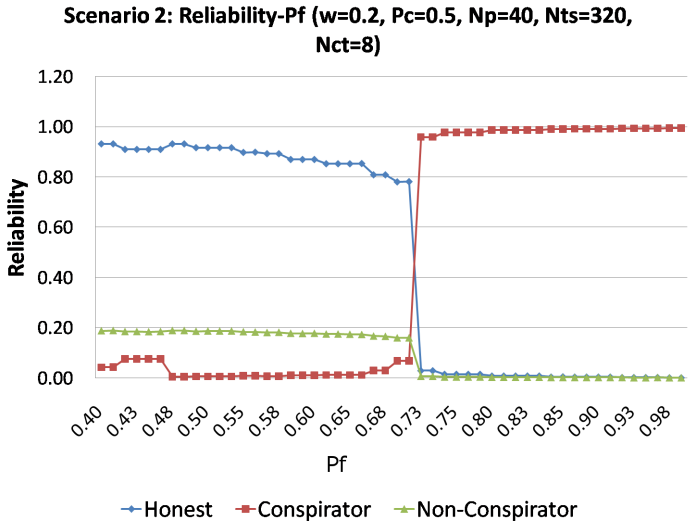


Figure 2.4: Changes in the reliabilities of three types of workers with the increase of P_f, in Scenarios 2.

situation, which corresponds to Equation (2.12). Once the malicious rate is high enough to exceed a certain threshold (70 percent in this example), which can be calculated by Equation (2.11), the average of the expected reliabilities of conspirators quickly overwhelms that of honest workers. This matches Equation (2.13) quite well.

Scenario 3

The evaluation results of Scenario 3 are similar with those of Scenario 2. An example of Scenario 3 is shown in Figure 2.5. The system size, task set size, and checking task size are 40, 320 and 8, respectively. The results show that, when P_f is low (lower than 0.73 in this case), the average of the expected reliabilities

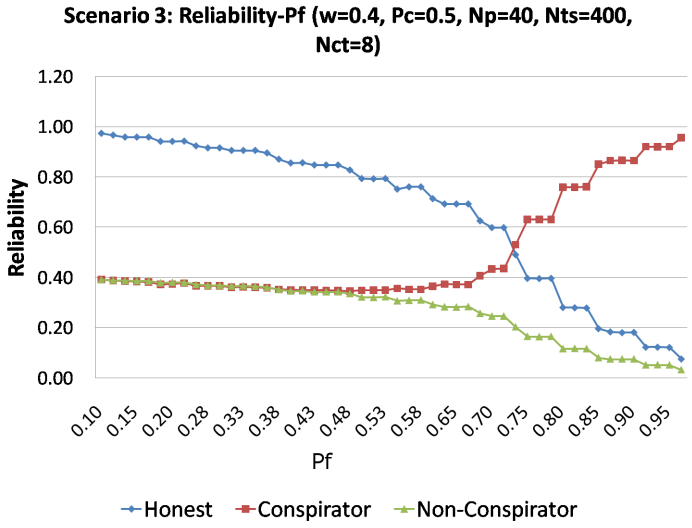


Figure 2.5: Changes in the reliabilities of three types of workers with the increase of P_f, in Scenarios 3.

of honest workers is higher than that of malicious workers, and the reliability gap is large and clear. As P_f increases, conspirators gradually overtake honest workers. The reason is that, as the number of malicious workers increases, conspirators also increase. As a result, conspirators can detect more checking tasks in their task sets. Reliabilities of honest workers decrease as well since they are more frequently paired with malicious workers, resulting in a higher miss-matching rate.

2.5.2 Convergence Performance

Figures 2.6 and 2.7 review the convergence performance of honest and malicious workers in Scenario 1, respectively. For instance, Figure 2.6 shows the maximal, minimal and average values of the reliabilities of all honest workers during the running of the system. This reveals that the average reliability converges in about ten rounds. Moreover, the reliabilities of all honest workers in each round

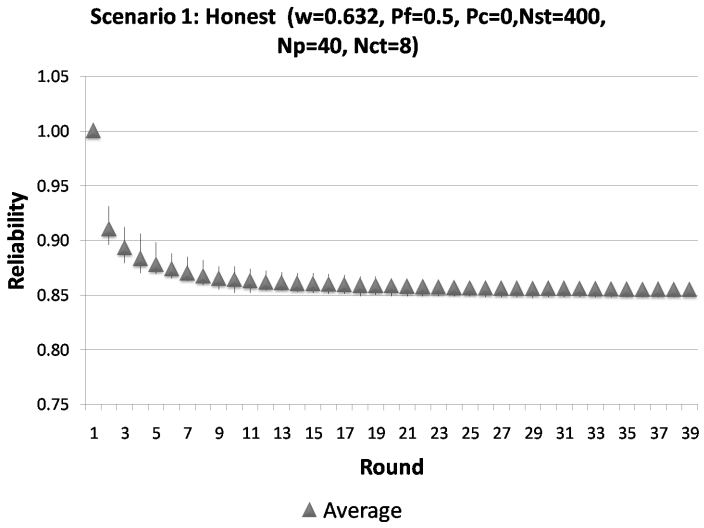


Figure 2.6: The convergence performance of honest workers during the system running in Scenario 1.

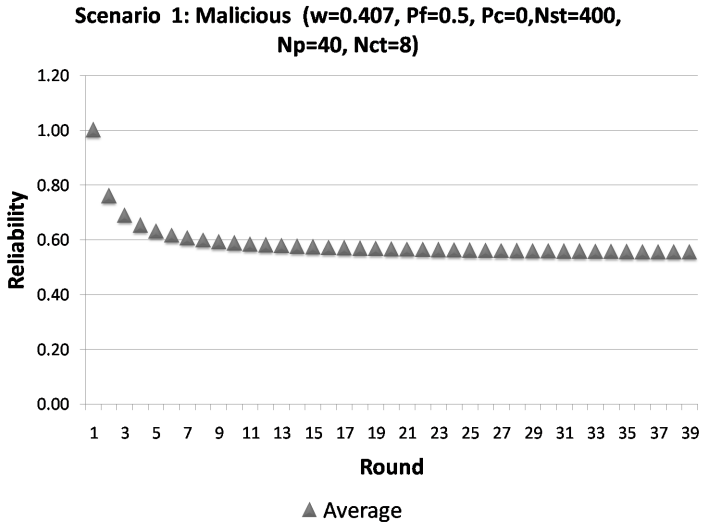


Figure 2.7: The convergence performance of conspirators during the system running in Scenario 1.

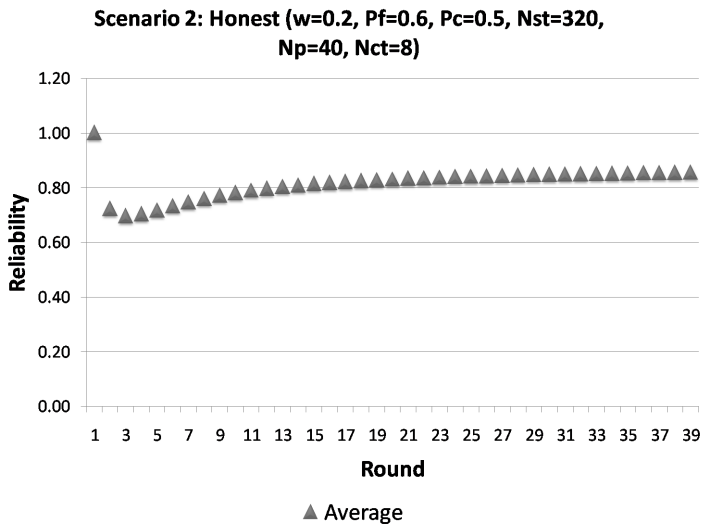


Figure 2.8: The convergence performance of honest workers during the system running in Scenario 2.

stay within a small range (less than 5% in this example) around the average. This means that it is unlikely for the reliability of a malicious worker to surge over that of an honest worker throughout the running of the system. Figures 2.8, 2.9 and 2.10 reveal the convergence status of each worker type in Scenario 2. These results are all similar to the results of Scenario 1.

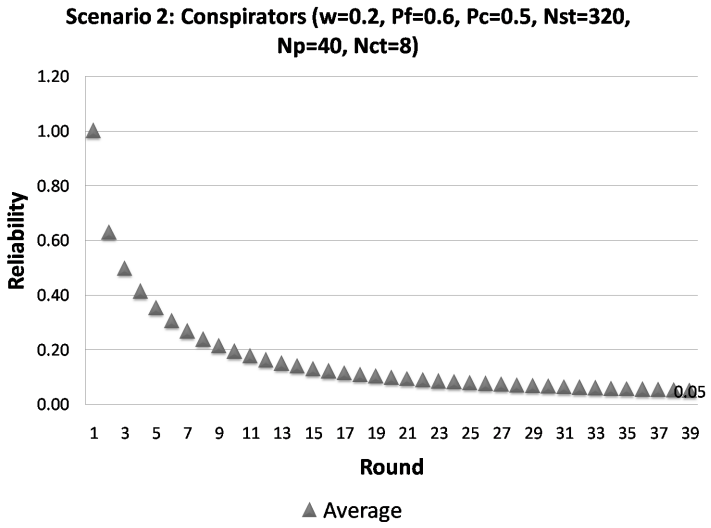


Figure 2.9: The convergence performance of conspirators during the system running in Scenario 2.

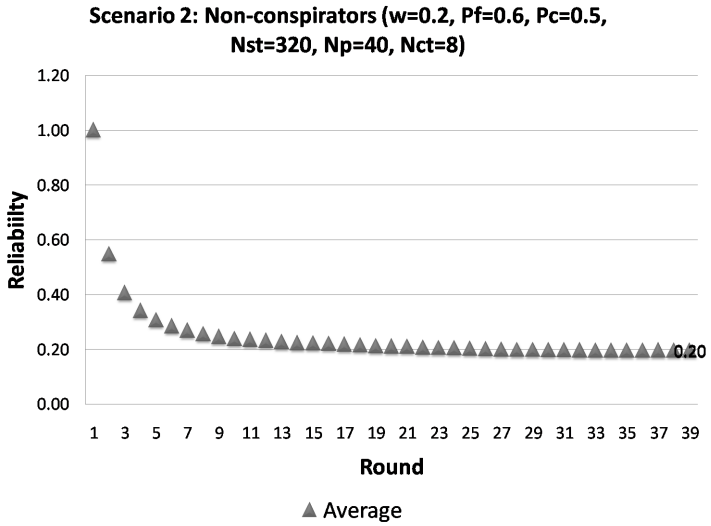


Figure 2.10: The convergence performance of non-conspirators during the system running in Scenario 2.

2.6 Conclusion

DCSs are vulnerable to the false result attack, and the existing mechanisms to this attack are either inefficient or impractical. This chapter proposed MSC, an algorithm that solved the problems of the existing solutions. MSC enables the host to detect malicious workers. The key idea of MSC is to use checking tasks, instead of quizzes, to detect malicious workers. Hence, MSC removes the need of generating quizzes. Theoretical analysis and simulation show that: in non-collusion DCSs, the host can always detect malicious workers; in collusion DCSs, as far as malicious workers are less than honest ones, the host can accurately detect malicious workers. Additionally, the theoretical efficiency of MSC can be near optimal.

Chapter 3

SybilDetector: an Attack Edge Detecting-Based Sybil Detecting Algorithm

3.1 Introduction

The Sybil attack is a serious security problem to DCSs. In a DCS, it is easy for a malicious user to create many malicious nodes. The malicious user can then control his/her malicious nodes to break the system protocol. This attack is called the Sybil attack, and malicious nodes in this attack are called Sybil nodes. For example, the Sybil attack can break the voting systems of large online communities like Amazon[30, 31]. In a voting system, each node has one ballot and can give its ballot to certain goods (such as a book). The goods that obtain more ballots will be advertised to more nodes. By controlling many Sybil nodes, malicious users can interrupt the voting results arbitrarily.

Among existing resisting mechanisms to the Sybil attack[32, 33], the social

network model (SNM) based Sybil detecting (SSD) algorithms (SybilLimit[18], Gatekeeper[17], SOHL[19]) are drawing enormous attention from researchers. For a DCS, SSD algorithms assume that DCSs obey SNM. In such a DCS, SSD algorithms enable each honest node to judge the types of other nodes. The basic idea of SSD algorithms is that, since the number of attack edges is small, the communication between nodes of different types is weakened. Hence, it is easier for honest nodes to communicate with honest nodes than with Sybil nodes. By utilizing this property, honest nodes can distinguish honest nodes from Sybil nodes.

The problem of existing SSD algorithms is that their accuracies are low – these algorithms have a low honest accept rate (har) or a high Sybil accept rate (sar). Although the attack edge bottleneck can weaken communication between nodes of different types, it cannot entirely stop the communication. Accordingly, it is possible for honest nodes to make incorrect judgments.

This dissertation observes that detecting the attack edges plays an important role in creating accurate SSD algorithms. Here, detecting attack edges means to enable nodes in the system to judge whether or not a certain group of edges are attack edges. Intuitively, by explicitly detecting the attack edges, it is possible to more clearly separate nodes of different types, and thus to create accurate SSD algorithms.

The objective of this chapter is to design accurate SSD algorithms for authorized DCSs by utilizing the attack edge detecting technique. This chapter designs SybilDetector, an attack edge detecting-based SSD algorithm for authorized DCS. In a SNM-based system, the shortest paths between honest nodes and Sybil nodes have to pass the attack edges. In SybilDetector, for each honest node hn, hn detects the attack edges in the system. Then, for node u, hn judges

whether the shortest paths between it and u pass the detected edges. If and only if it is true, hn regards u to be Sybil (See Figure 1.5 for intuition). The core of SybilDetector is a mechanism that enables each honest node to judge whether or not a certain edge is an attack edge. The performance of SybilDetector is compared with that of an existing SSD algorithm by simulations on topologies of synthetic and real world networks. Evaluation results show that the sar of SybilDetector is considerably lower than that of SybilLimit. Additionally, this chapter validates the potential of the attack edge technique in designing accurate SSD algorithms.

Model and Denotations

SybilDetector assumes that the system obeys SNM. The number of honest nodes, denoted by n, is known to each node, as many existing SSD algorithms do[17, 20]. The number of attack edges in the system, denoted by g, is $o(n/\log n)$*. The reason of this assumption is explained in Section 3.2.3. The number of Sybil nodes in the system, denoted by snn, is $O(n)$. The edges among honest nodes are called the *honest edges*, and the number of honest edges is denoted by m. Hernando et al.[35] has shown that, in social networks, the number of friends a person has is on average 150. Hence, SybilDetector considers that $m = O(n)$. The *diameter* of the system, denoted by $\triangle$, is assumed to be $O(\log n)$[36]. Here, the diameter of a network system is the maximum of the distances between any pair of nodes in the system. Table 3.1 lists the important denotations used in this chapter.

*In computing complexity theory[34], the asymptotic notations $f(n) = O(g(n))$ means that, as n increases, $\exists k, |f(n)| < k \cdot g(n)$. $f(n) = \Omega(g(n))$ means that, as n increases, $\exists k, f(n) > k \cdot g(n)$. $f(n) = \Theta(g(n))$ means that, as n increases, $\exists k_1, \exists k_2, g(n) \cdot k_1 \leq f(n) \leq g(n) \cdot k_2$. $f(n) = o(g(n))$ means that, as n increases, $\forall \varepsilon, |f(n)| < \varepsilon \cdot g(n)$.

Table 3.1: Important denotations

Name	Description
(s,t)-SPs	The shortest paths between node s and node t
First-type SPs	The shortest paths between nodes of different types
Second-type SPs	The shortest paths between honest nodes
n	The number of honest nodes in the system
snn	The number of Sybil nodes in the system
Honest edges	The edges among honest nodes
m	The number of honest edges in the system
har	The probability that node v accepts node u, where v and u are two random honest nodes
sar	The probability that node v accepts node u, where v and u are a random honest node and a random Sybil node, respectively
Benchmark suspects of v	The set of random honest nodes sampled by node v to estimate its bottleneck bound
$\aleph$	The number of benchmark suspects sampled per node
SPEB	The shortest path edge betweenness

The following part of this chapter is organized as follows[†]. Section 3.2 provides more discussion on the Sybil attack and SSD algorithms. Section 3.3 details the design of SybilDetector. Section 3.4 evaluates the performance of SybilDetector. Then, Section 3.5 discusses the potential problems of SybilDetector. Finally, Section 3.6 concludes this chapter.

[†]This chapter is an extended version of [37].

3.2 Related Work

3.2.1 Sybil Attack Resisting Mechanisms

The Sybil attack was first noticed by Douceur[12] in the research of P2P systems. Douceur found that, in P2P systems, it is easy for a user to control a large number of nodes to breach the system protocols. After the research of Douceus, the Sybil attack rapidly gained the attention of researchers of not only P2P systems, but also many other DCSs such as sensor network systems[16] and mobile systems[38].

Many Sybil resisting solutions have been proposed[33], and Levine et al.[33] extracted the two basic ideas of these solutions. Each idea has its advantages and disadvantages, and no complete solution to the Sybil attack has yet been provided.

The first idea is to ensure the identifications of nodes using trustful authorities. Douceur proved that, in a DCS with no trustful authority, it is impossible to address the Sybil attack completely[12]. Trustful authorities can impose stricter control on the identifications of nodes. This makes it harder for malicious users to create Sybil nodes and thus weakens the attack.

The second idea is to identify Sybil nodes by testing the resources held by each node. In a DCS, suppose that each user holds finite resources such as memory, bandwidth and computing power. If many Sybil nodes are created, the resources per Sybil node will be notably less than resources per honest node. This property can be used to distinguish Sybil nodes from honest ones. SNM-based Sybil resisting algorithms are indeed based on this idea, where the edges incident to honest nodes are the limited resources.

3.2.2 Social Network Model

SNM is a model that depicts the network topologies of DCSs. For a DCS, SNM assumes that the number of attack edges in the system is small. Here, an attack edge is an edge connecting nodes of different types, honest or Sybil. The attack edges separate the whole network into two regions, the honest region of honest nodes and the Sybil region of Sybil nodes.

SNM fits systems that contain trust relationships. Online social network systems (e.g., Facebook[1]) are the representative systems that match SNM. In such a system, each node is corresponding to a person, and edges among the nodes represent friendship among the people. Two nodes are connected by an edge usually because the corresponding users of these two nodes trust each other in the real world. It is reasonable to consider that, in the real world, it is hard for a malicious person to be trusted by many honest persons. Consequently, in online social network systems, the number of attack edges should be small.

Although not all DCSs obey SNM, SNM-based security mechanisms for DCSs still have attracted great attention of researchers. Some DCSs naturally obey SNM. For example, in many P2P systems, for a new node to join the system, the new node needs to obtain invitations of the nodes that already exist in the system[14]. Having joined the system, the new node is connected to the nodes that have invited it. It is reasonable to think that honest nodes are unlikely to invite Sybil nodes. Hence, the network topologies of these P2P systems naturally obey SNM, and these DCSs can readily benefit from SNM-based security mechanisms. For DCSs that do not obey SNM, they can import social network information from third party online social network systems to create security mechanisms[39]. As the potential of SNM in consolidating the security of DCSs is explored, it is expected that more DCSs will be constructed based on SNM.

3.2.3 SNM-Based Sybil Detecting Algorithms

This section introduces some representative SSD algorithms.

SybilLimit[18]: SybilLimit is one of the earliest and fundamental SSD algorithms. SybilLimit created the *probing random walk*, which has become a core constructing component for many SSD algorithms. A probing random walk is a message packet that advances for $O(\log n)$ steps in a random walk manner. Yu el at.[18] proved that, when $g = o(\frac{n}{\log(n)})$, the probing random walk has two good properties:

- First, the escape rate of the probing random walk is low of $o(1)$. Here, the escape rate of the probing random walk is the average probability that a probing random walk starting from the honest region enters the Sybil region during its movement.

- Second, for each probing random walk starting from the honest region, in the last step, this probing random walk traverses each honest edge with an equal probability of $1/m$. Specifically, In a DCS with m edges, let rw be a random walk and e be an arbitrary edge. According to Markov theory, once rw has moved sufficient steps, the probability that rw passes e from either direction converges to $1/(2m)$[40]. At this point, it is said that rw has *mixed*. The number of steps for rw to mix changes according to the underlying network. A network where random walks mix within $\log n$ steps is said to be *fast mixing*[18]. Yu el at.[18] showed that social networks are fast mixing. For a SNM-based DCS, let prw be a probing random walk starting from the honest region. Within $\log n$ steps, prw stays within the honest region and fixes within the honest region. Hence, the last step of prw traverses each honest edge with the probability of $1/m$.

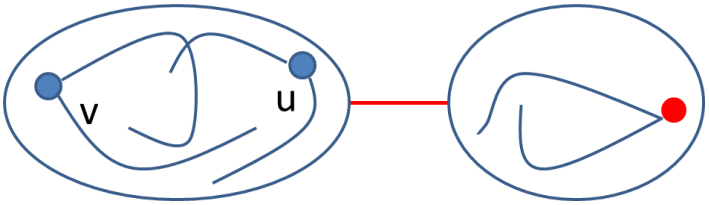


Figure 3.1: SybilLimit

In SybilLimit, each node disseminates $r = O(\sqrt{m})$ probing random walks. Two nodes accept each other if and only if their respective probing random walks intersect. Since the escape rate of the probing random walk is low, the probability that probing random walks of nodes of different types intersect is low. Accordingly, honest nodes can distinguish honest nodes from Sybil nodes. Theoretically, in SybilLimit, each honest node accepts all other honest nodes with a high probability and accepts $O(g \log n)$ Sybil nodes. Figure 3.1 shows an example of SybilLimit.

SOHL[19] and **Whānau**[20]: SOHL and Whānau aim to reduce the message cost of SybilLimit. In SybilLimit, the message cost for node v to judge the type of node u is $O(\sqrt{m})$. SOHL and Whānau reduce this cost to $O(1)$.

Similar to SybilLimit, the core constructing technique of SOHL and Whānau is the probing random walk. In SOHL and Whānau, each node v disseminates a large number of probing random walks. Then, v accepts the ending nodes of its probing random walks and rejects other nodes. Since the escape rate of the probing random walk is low, honest nodes can distinguish honest nodes from Sybil nodes.

Sumup[31]: In Sumup, each node v disseminates a certain number of *tickets* to other nodes in a broad first search manner. On receiving a set of tickets, node u takes one ticket, and sends the leftover tickets to its incident nodes that have not yet received any ticket. Finally, for an unknown node u, v accepts u if and

only if u has received a ticket of v. Since the number of attack edges is small, the probability that the tickets of v enter the Sybil region is low. Accordingly, v can distinguish Sybil nodes from honest ones.

Gatekeeper[17]: Gatekeeper increases the accuracy of Sumup using a *majority vote technique*. As in Sumup, in Gatekeeper, each node v disseminates a certain number of tickets in a broad first search manner. v regards the nodes that have received its tickets as its *reachable* nodes. Then, v accepts node u if and only if u is reachable to multiple random honest nodes. When u is Sybil, the probability that u is reachable to any honest node is low. Hence, the probability that u is reachable to multiple honest nodes is even lower. In this way, Gatekeeper achieves a lower sar than Sumup. Theoretically, Gatekeeper enables each honest node to accept all other honest nodes, and accept $O(\log g)$ Sybil nodes.

Although these SSD algorithms have designed various techniques to distinguish Sybil nodes, they have not tried to explicitly detect the attack edges.

3.2.4 Betweenness Metrics

For a DCS, a *node betweenness metric* (*edge betweenness metric*) is a metric that measures the extent to which each node (edge) lies on message paths between nodes[41]. The problem of detecting attack edges in SNM-based DSCs is relevant to the problem of computing the betweennesses of edges, which is discussed in Section 3.2.5. For a DCS, according to the way to transmit the message, so far, three kinds of node betweenness metrics and two kinds of edge betweenness have been defined.

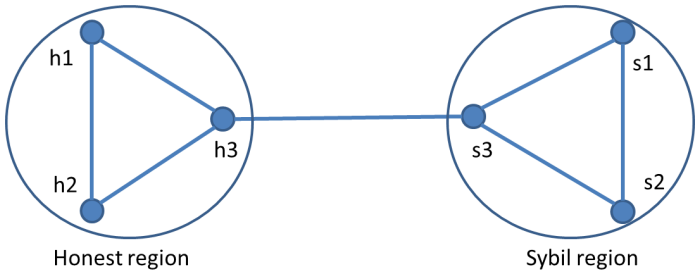


Figure 3.2: Attack edges have higher betweennesses

Shortest path node betweenness (**SPNB**)[42]: suppose that messages are transmitted along the shortest paths between nodes. The SPNB of each node v represents the fraction of messages that pass v when a message is transmitted from nodes s to t, average over all s and t.

Shortest path edge betweenness (**SPEB**)[43]: suppose that messages are transmitted along the shortest paths between nodes. The SPEB of each edge e represents the number of messages that pass e when a message is transmitted from nodes s to t, for all s and t. Newman et al.[43] observed that the SPEB satisfies the following *detecting property*: on average, the SPEBs of inter-cluster edges are higher than those of other edges. Figure 3.2 provides an example showing this property: the inter-cluster edge $(h3, s3)$ has a SPEB of 18, which is the highest among SPEBs of all edges.

Flow node betweenness (**FNB**)[44]: suppose that each edge in the system has a unit capacity. The flow node betweenness of a node v is the amount of message flow that passes v when the maximum flow is transmitted from nodes s to t, averaged over all s and t.

Random walk node / edge betweenness (**RWNB/RWEB**)[41]: suppose that messages are transmitted in a random walk manner between nodes: each message starts from its source, moves in a random walk manner until it reaches its destination. The RWNB of each node v (the RWEB of each edge e) represents the

fraction of messages that pass v (e) when a message is transmitted from nodes s to t, averaged over all s and t. Newman[41] showed that, in a DCS, the RWNBs of *front nodes* (nodes connecting multiple clusters) are higher than those of other nodes.

3.2.5 Sybil Resisting Network Clustering

The SSD algorithm proposed in this chapter is an extension of SRNC[37] – the only existing SSD algorithm that aims to detect the attack edges. This section gives a brief introduction to SRNC.

In SRNC, each honest node hn detects the attack edges in the system. Then, hn regards node u as a Sybil node if the shortest paths between hn and u pass the attack edges. To simplify the expression, the set of shortest paths between nodes s and t are written as (s,t)-SPs. The shortest paths between nodes of different types are called the *first-type shortest paths*, and the shortest paths between honest nodes are called the *second-type shortest paths*. The observation of SRNC is that, if u is Sybil, each of the (hv,u)-SPs has to pass at least one attack edge. In contrast, if u is honest, most (hn,u)-SPs should stay within the honest region and not pass any attack edges (See Figure 3.2 for intuition).

To detect the attack edges, a three-step scheme has been proposed to design attack edge detecting algorithms[37]:

- First, design an (or choose an existing) edge betweenness metric that satisfies the detecting property. This dissertation calls such a metric a *detecting metric*.
- Design an algorithm that enables each node to securely compute the betweennesses of edges in a distributed manner.

- Finally, design an algorithm that enables each node v to compute a *detecting threshold* that is higher than the average betweenness of honest edges, and lower than the average betweenness of attack edges. Then, v regards edges with betweennesses higher than the detecting threshold as attack edges.

Based on this scheme, the following *SPEB-based attack edge detecting* (SPEB-AED) scheme was further designed for creating SPEB-based attack edge detecting algorithms[37]. First, choose the SPEB as the detecting metric. Then, compute the SPEBs of edges using distributed shortest path computing algorithms. Finally, compute the average of the SPEBs of honest edges as the detecting threshold.

Note that the SPEB-AED scheme can only be implemented in authorized DCSs. Under SPEB-AED, nodes need to compute the shortest path information, which is a non-trivial problem in DCSs containing malicious nodes. Several algorithms, such as SEAD[45] and S-RIP[46], that can securely compute the shortest path information among honest nodes have been proposed. However, these algorithms require the support of trustful authorities. Therefore, the SPEB-AED scheme can only be implemented in authorized DCSs.

To detect the attack edges, SRNC tried to implement the SPEB-AED scheme. However, SRNC failed to find a way to compute the detecting threshold, and left this problem open.

3.3 SybilDetector

This section details the design of SybilDetector. Specifically, for an authorized DCS, SybilDetector enables each honest node hn to judge the other nodes in the system. The basic idea of SybilDetector is the same as SRNC: hn first detects the attack edges. Then, hn regards node u to be Sybil if each of the (hn, u)-SPs passes at least one attack edge. Additionally, SybilDetector detects attack edges using the SPEB-AED scheme. Different from SRNC, SybilDetector can compute the detecting threshold using a *benchmark technique*, which is introduced in Section 3.3.3. SybilDetector has three main steps.

- Step 1: hn computes the shortest paths between itself and all other nodes.
- Step 2: For each (hn, u)-SP, denoted by sp, hn uses the maximum value of the betweennesses of the edges along sp as the *bottleneck* of sp, denoted by $hn.bn(sp)$. In the example DCS in Figure 3.2, let $sp1 =< h1, h3, s3, s1 >$ be a $(h1, s1)$-SP, and let $sp2 =< h1, h2 >$ be a $(h1, h2)$-SP. The bottleneck of $sp1$ is equal to the SPEB of edge $(h3, s3)$ (i.e., 18), while the bottleneck of $sp2$ is equal to the SPEB of edge $(h1, h2)$ (i.e., two).
- Step 3: hn computes a *bottleneck bound*, denoted by $hn.bb$. Then, hn regards a node u to be Sybil if and only if the bottleneck of each (hn, u)-SP is larger than $hn.bb$.

Sections 3.3.1, 3.3.2 and 3.3.3 detail each step, respectively.

3.3.1 Compute the Shortest Paths

First, nodes run SEAD[45] to compute information of the shortest paths. As introduced in Section 3.2.5, SEAD ensures that honest nodes compute the correct

shortest paths between each other in a distributed manner under malicious interference. Sybil nodes may not participate in the computing of shortest paths. If so, honest nodes cannot find shortest paths to the Sybil nodes and thus cannot compute the correct betweennesses of attack edges. Consequently, honest nodes cannot identify Sybil nodes. To address this problem, after the computing of shortest paths, hn accepts node u only if hn has found at least one (hn, u)-SP.

3.3.2 Compute the Bottlenecks of Shortest Paths

Then, hn computes the bottlenecks of the shortest paths between itself and other nodes. First, for each incident edge e, hn computes the betweenness of e, denoted by $hn.b(e)$, as the number of the shortest paths that pass e. Then, let $sp =< u = v_1, v_2, ..., v_w = hn >$ be a (u, hn)-SP. To know the bottleneck of sp, hn asks each node v_i along sp for the betweenness of edge (v_i, v_{i+1}) $(0 \leq i < w)$. Finally, hn uses the maximum value of the received betweennesses as the bottleneck of sp.

3.3.3 Compute the Bottleneck Bound

Now, hn needs to compute its bottleneck bound. The bottleneck bound should be large, so that most honest nodes can be accepted. However, the bottleneck bound should be small, so that most Sybil nodes can be rejected. Hence, the value of the bottleneck bound has a trade-off between har and sar.

Specifically, hn computes a bottleneck bound so that hn can accept β percent of honest nodes. Here, β is a parameter used to adjust the trade-off between har and sar. In SRNC, the bottleneck bound of each node is set to be a global parameter. In SybilDetector, hn can compute its bottleneck bound locally, using the following benchmark technique.

1. hn finds a set of א honest nodes from the systems as its *benchmark suspects*,

denoted by $hn.bs$. hn uses ***MH-random walks*** to find these honest nodes[17] (Section V.B). A MH-random walk is a message packet that moves $\log n$ hops in a special random walk manner. Suppose that the walk is currently on node i, i chooses an incident node j as the next hop with a probability of $\min(1/d_i, 1/d_j)$, where d_i is the number of incident nodes of i. As discussed by Tran et al.[17], when $g = o(n/\log n)$, a MH-random walk starting from the honest region will stay within the honest region, and finally end on a random honest node with a high probability.

2. Call the average of the bottlenecks of the (hn, u)-SPs the ***bottleneck of*** u ***for*** hn, denoted by $hn.bn(u)$. hn sorts its benchmark suspects so that

$$hn.bn(bs_0) < ... < hn.bn(bs_i) < ... < hn.bn(bs_{\aleph}), \forall bs_i \in hn.bs. \tag{3.1}$$

3. hn uses $hn.bn(bs_{\beta \cdot \aleph})$ as its bottleneck bound.

The idea behind this benchmark technique is as follows. Obviously, the bottleneck bound chosen by the above process allows hn to accept β percent of the benchmark suspects. The benchmark suspects are random samples of honest nodes. Therefore, if a bottleneck bound ensures hn to accept β percent of the benchmark suspects, it should also ensure hn to accept β percent of all honest nodes.

Having obtained the information relating to shortest paths, bottlenecks of shortest paths, and the bottleneck bound, hn is able to determine which nodes to accept. For node u, hn accepts u if and only if 1). hn has found at least one shortest path to u, and 2). at least one (hn, u)-SP, sp, satisfies that $hn.bn(sp) < hn.bb$.

3.3.4 Overhead

The total message overhead of SybilDetector is $O(n^2 \log n)$, which is equal to the overheads of SybilLimit[18] and Gatekeeper[17].

As described in the previous section, SybilDetector has three steps: 1). computing the information relating to the shortest paths, 2). computing the bottlenecks, and 3). computing the bottleneck bounds. In the first step, SybilDetector uses SEAD to compute the shortest paths. SEAD is a distance vector protocol-based algorithm[47]. The message cost of distance vector protocol-based shortest path computing algorithms is $O(mn\triangle)$, when they run in a synchronous mode[37]. According to the assumptions made in Section 3.1, $O(mn\triangle) = O(n^2 \log n)$. Therefore, the message cost of SybilDetector for computing shortest paths is $O(n^2 \log n)$.

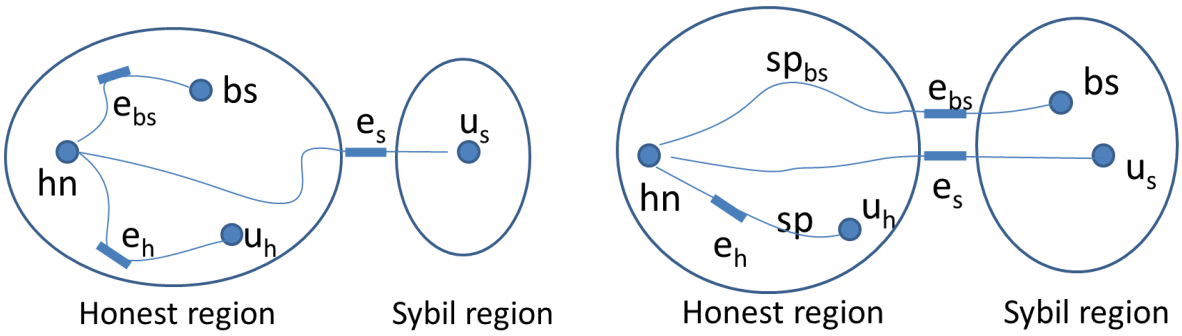
In the second step, for each node hn and each (hn, x)-SP, sp, to compute the bottlenecks of sp, hn asks each node along sp for betweennesses of the edges along sp. Hence, in this step, the message overhead is $O(n \log n)$ for hn and is $O(n^2 \log n)$ for the whole system.

In the third step, for node hn, hn needs to sample $\aleph$ benchmark suspects, where each sampling has a message overhead of $O(\log n)$. As shown in Section 3.4, $\aleph$ can be a small constant like 50 in practice. Therefore, the message overhead of the whole system in this step is $O(n \log n)$.

In conclusion, the message cost of the whole system is $O(n^2 \log n)$ and is $O(n \log n)$ per node.

3.3.5 Analysis

Intuitively, the performance of SybilDetector (har and sar) is mainly affected by g and snn. This section provides a qualitative analysis of the influence of g and



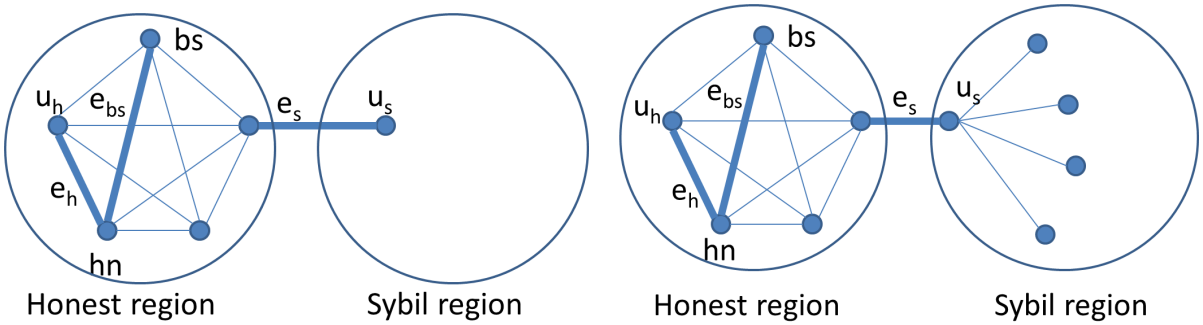
(a) g is small.

(b) g is large.

Figure 3.3: Influence of g on har and sar

Table 3.2: Changes as g increases (corresponding to Figure 3.3)

Parameter	Change	Reason
$b(e_h)$	Does not change	There is no change in the number of shortest paths in the system. Additionally, the change of g does not affect the network topology of the honest region. Therefore, the change of g does not affect $b(e_h)$.
$b(e_s)$	Decreases	e_s is an attack edge. As g increases, the average number of shortest paths passing each attack edge decreases. Accordingly, $b(e_s)$ decreases.
$b(e_{bs})$	May not change	As g increases, the probability that bs is Sybil and the probability that e_{bs} is an attack edge increase. The attack edges have high betweennesses. Therefore, $b(e_{bs})$ should increase. However, as g increases, the average betweenness of attack edges decreases. Therefore, as g increases, har may not change.
har	May not change	$har = Pr(b(e_h) < b(e_{bs}))$. $b(e_h)$ does not change while $b(e_{bs})$ may not change. Therefore, har may not change.
sar	Increases	$sar = Pr(b(e_s) < b(e_{bs}))$. $b(e_{bs})$ may not change, while $b(e_s)$ decreases. Accordingly, sar increases.



(a) snn is small.

(b) snn is large.

Figure 3.4: Influence of snn on har and sar

Table 3.3: Changes as snn increases (corresponding to Figure 3.4)

Parameter	Change	Reason
$b(e_h)$	Does not change or increase	As snn increases, the number of first-type shortest paths increases. Therefore, $b(e_h)$ will increase if e_h is along the increased shortest paths. Otherwise, $b(e_h)$ does not change.
$b(e_s)$	Increases	e_s is an attack edge. As snn increases, the number of first-type shortest paths increases. Accordingly, the betweennesses of attack edges increase, and $b(e_s)$ increases.
$b(e_{bs})$	Does not change or increase	The number of first-type shortest paths increases. $b(e_{bs})$ increases if e_{bs} is on these increased shortest paths. Otherwise, $b(e_{bs})$ does not change.
har	Does not change	Both bs and u are random honest nodes. Therefore, $b(e_h)$ and $b(e_{bs})$ change in the same way. Accordingly, $har = Pr(b(e_h) < b(e_{bs}))$ does not change.
sar	Does not change or decrease	$b(e_s)$ increases while $b(e_{bs})$ may not change. Accordingly, $sar = Pr(b(e_s) < b(e_{bs}))$ should stay unchanged or decrease.

snn on performance. It is shown that, as g increases, har may not change, and sar increases. As snn increases, har does not change, and sar decreases.

First, the following analysis simplifies the expressions of har and sar. See Figure 3.3 and Figure 3.4 for intuition of this analysis.

- Let hn be an average honest node. Additionally, let u_h (u_s) be another random honest (Sybil) node. According to its definition, har (sar) is equal to the probability that hn accepts u_h (u_s).

- Let sp_h be an average (hn, u_h)-SP, and let sp_s be an average (hn, u_s)-SP. Then, the probability that hn accepts u_h is equal to $Pr(hn.bn(sp_h) < hn.bb)$, and the probability that hn accepts u_s is equal to $Pr(hn.bn(sp_s) < hn.bb)$. Here, $Pr(\chi)$ means the probability that χ happens.

- Let bs be the benchmark suspect that determines the bottleneck bound of hn (i.e., bs is the benchmark suspect $bs_{\beta \cdot \aleph}$ in Equation (3.1)). Additionally, let sp_{bs} be an average (hn, bs)-SP. Then, $hn.bb = hn.bn(sp_{bs})$.

- Let e_h be the edge along sp_h that decides $hn.bn(sp_h)$ (i.e., e_h is the edge along sp_h that has the largest betweenness). Similarly, let e_s be the edge along sp_s that decides $hn.bn(sp_s)$, and let e_{bs} be the edge along sp_{bs} that decides $hn.bn(e_{bs})$. Then, $hn.bn(sp_{bs}) = b(e_{bs})$, $hn.bn(sp_h) = b(e_h)$ and $hn.bn(sp_s) = b(e_s)$, where $b(e)$ represents the SPEB of edge e.

In brief, the above analysis shows that $har = Pr(b(e_h) < b(e_{bs}))$ and $sar = Pr(b(e_s) < b(e_{bs}))$.

Tables 3.2 and 3.3 summarize the changes of har and sar as g and snn increase, respectively. For example, Table 3.2 shows that, as g increases, har may not change:

- As g increases, $b(e_h)$ does not change. There is no change in the number of shortest paths in the system. Besides, the increase of attack edges does not affect the network topology of the honest region. Accordingly, $b(e_h)$ should have no significant change.

- As g increases, $b(e_{bs})$ may not change. Recall that bs is sampled using MH-random walks. As g increases, the probability that MH-random walks of hn enters the Sybil region, and the probability that e_{bs} is an attack edge increases. Hence, $b(e_{bs})$ should increase. However, as g increases, the average betweenness of attack edges decreases. Therefore, b_{es} may not change.

- Therefore, as g increases, $har = Pr(b(e_h) < b(e_{bs}))$ may not change.

Detailed analysis of the influence of g and snn on har and sar are listed in Tables 3.2 and 3.3. These results are to be validated by simulation in Section 3.4.

3.4 Evaluation

This section evaluates the performance of SybilDetector using simulation. Specifically, this section has two goals.

The first goal is to evaluate the accuracy of SybilDetector. To this end, this section compares the accuracy of SybilDetector with that of SybilLimit. It is expected that the sar of SybilDetector is lower than that of SybilLimit. For a DCS, let hn be an honest node and u be a Sybil node. SybilDetector and SybilLimit are similar in the following way: in both SybilDetector and SybilLimit, for hn to accept sn, there must be a path connecting hn and sn (a shortest path for SybilLimit, and a random walk path for SybilLimit). Suppose that such a path p exists. Then, in SybilLimit, hn will accept sn. In contrast, in SybilDetector, hn will reject sn because hn knows that p has passed attack edges. Accordingly, the sar of SybilDetector should be lower than that of SybilLimit.

The second goal is to evaluate the influence of g and snn on the performance of SybilDetector. Section 3.3.5 qualitatively analyzed the influence of g and snn on the performance of SybilDetector. This section thus aims to validate these results.

3.4.1 Evaluation Configuration

Each network used in the simulations is created in the following way. First, the networks of the honest region and the Sybil region are created separately. Then, the two regions are connected by g attack edges. Two networks are used to create the honest region. One is a real world social network that represents the hyperlinks among the blogs created during the 2005 U.S. election[48]. This network is used as it represents a complete social network. Another network is

Table 3.4: Networks used for creating honest and Sybil regions

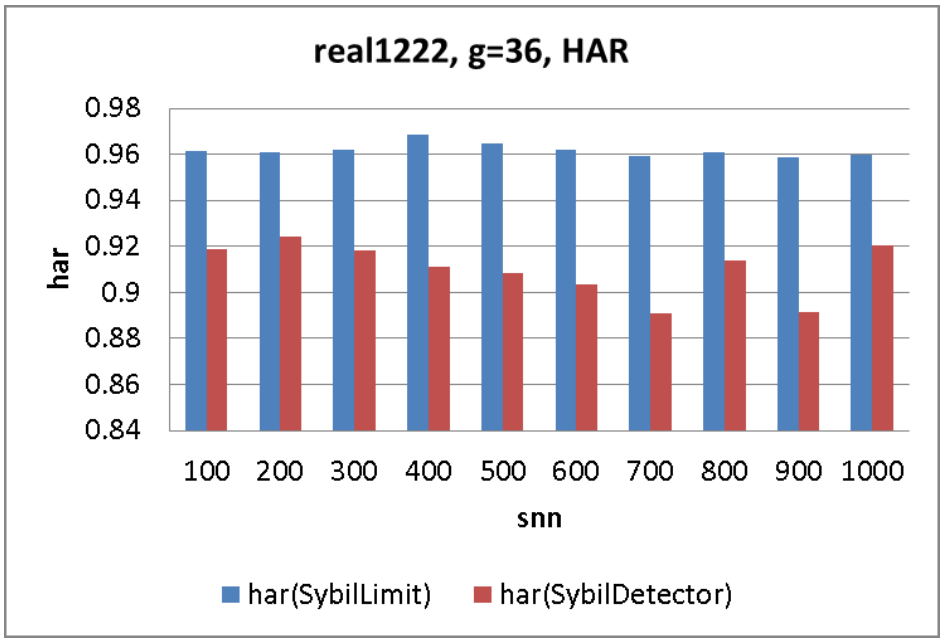
Network name	Type	Number of nodes	Number of edges
real1222	Real world social network	1,222	16,714
pl1222	Synthetic network of Barabasi-Albert model	1,222	7,257
rn500	Synthetic network of Erdos-Renyi model	500	1,725

Table 3.5: Networks used for evaluation

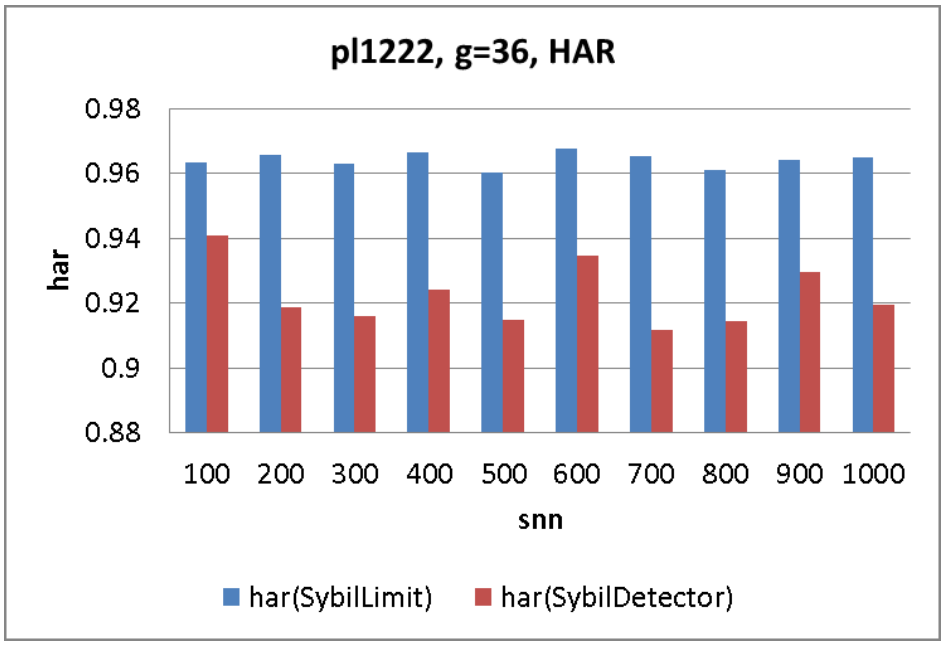
Network name	Honest region	Sybil region
G1	real1222	rn500
G2	pl1222	rn500

a synthetic network generated according to the Barabasi-Albert (BA) model[49] using NetworkX[50]. The BA model is designed for representing real world social networks. The networks of the Sybil regions are created using the Erdos-Renyi (ER) model[51]. In a network obeying the ER model, each pair of two nodes are connected by an edge with a probability of p. It is impossible to presume the network topology of a Sybil region because Sybil nodes can randomly change the connection between each other. Hence, this evaluation simply uses the ER model to create the Sybil regions. In this evaluation, p is set to 0.05. Table 3.4 lists the details of the networks used to create the honest region and Sybil region. Using these networks, two networks G1 and G2 are created for simulations, as listed in Table 3.5.

During each simulation, β for both SybilDetector and SybilLimit is 95%. This dissertation has tested different $\aleph$ ranging from 10 to 100, and found that the performance of SybilDetector is not sensitive to the change of $\aleph$. Hence, during the simulations, $\aleph$ is set to 50.



(a) G1



(b) G2

Figure 3.5: Changes of har as snn increases

3.4.2 Results and Analysis

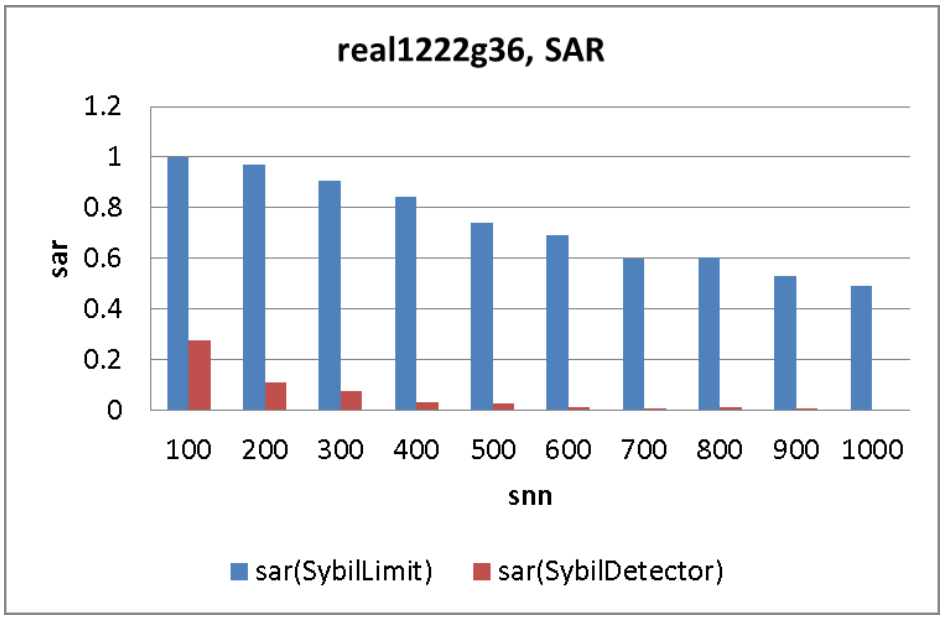
Figure 3.5 shows the changes of har as snn increases. g is 36 in these simulations. First, in both the real world social network and the synthetic network, the har of SybilDetector is about 92%, which is close to β. This indicates that the benchmark technique works effectively. Second, snn does not affect har, which matches the analysis in Section 3.3.5.

Figure 3.6 shows the changes of sar as snn increases. g is 36 in these simulations. Two trends are notable. First, as snn increases, the sar of SybilDetector decreases. This change is also explained in Section 3.3.5. Intuitively, as snn increases, the betweennesses of attack edges increase, as well. Hence, it is easier for honest nodes to detect the attack edges and Sybil nodes, and sar decreases. Second, the sar of SybilLimit is notably (at least 5x) higher than the sar of SybilDetector in both the real world and the synthetic network topologies.

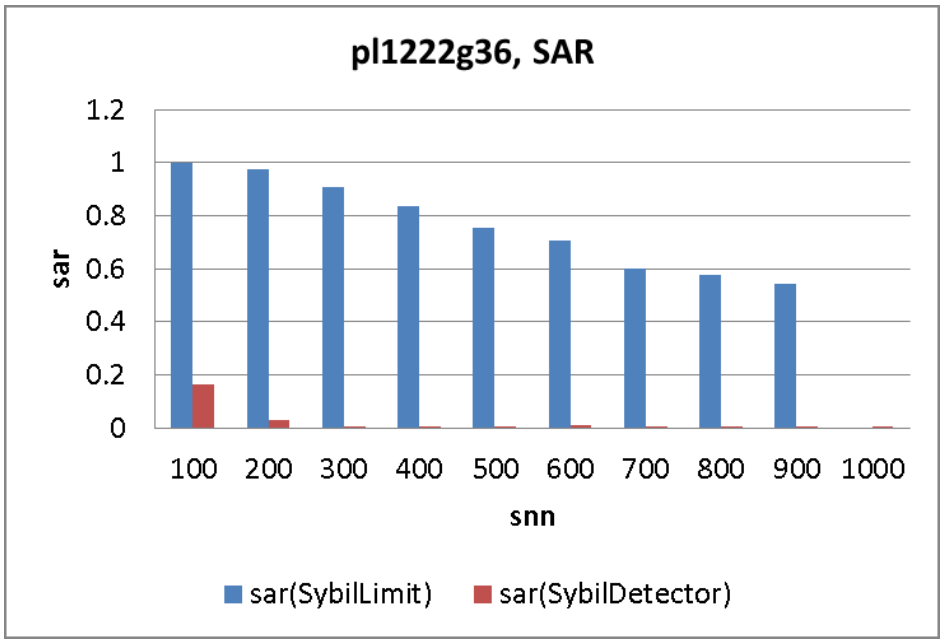
Figure 3.7 shows the changes of har as g increases. snn is 500 in these simulations. First, similar to Figure 3.5, the har of SybilDetector is close to β, which validates the effectiveness of the benchmark technique. Second, as g increases, the har of SybilDetector does not show obvious change, which matches the analysis of Section 3.3.5.

In both Figures 3.5 and 3.7, the har of SybilDetector is lower than the har of SybilLimit. However, the difference is slight and less than 5%.

Figure 3.8 shows the changes of sar as g increases. snn is 500 in each simulation. As g increases, the sar of both SybilLimit and SybilDetector increases. For SybilDetector, this is reasonable. As g increases, the betweennesses of the attack edges decrease, which makes it harder for honest nodes to detect the attack edges. Accordingly, the sar increases. However, the sar increases much faster in SybilLimit than in SybilDetector. For example, in both the real world

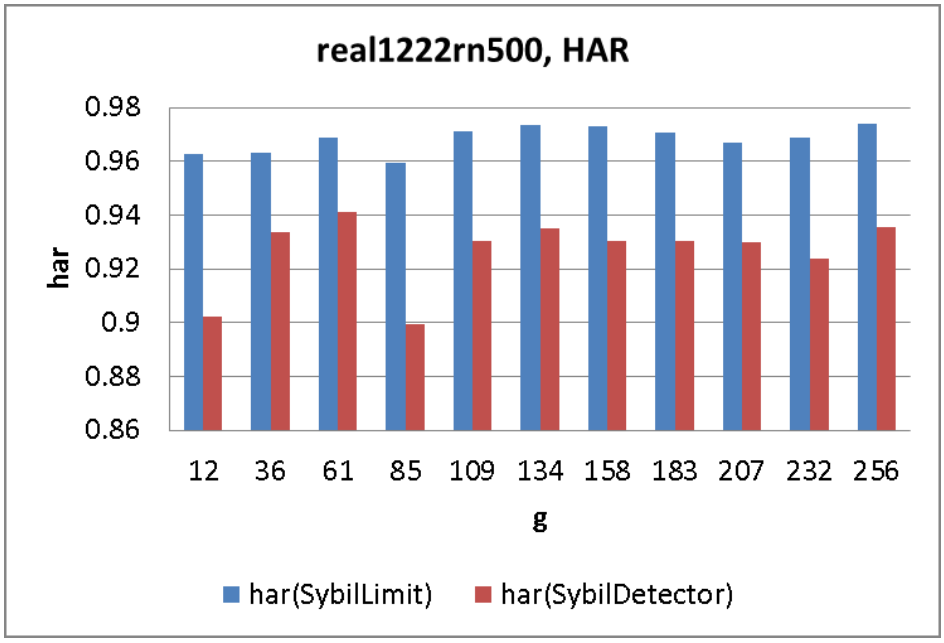


(a) G1

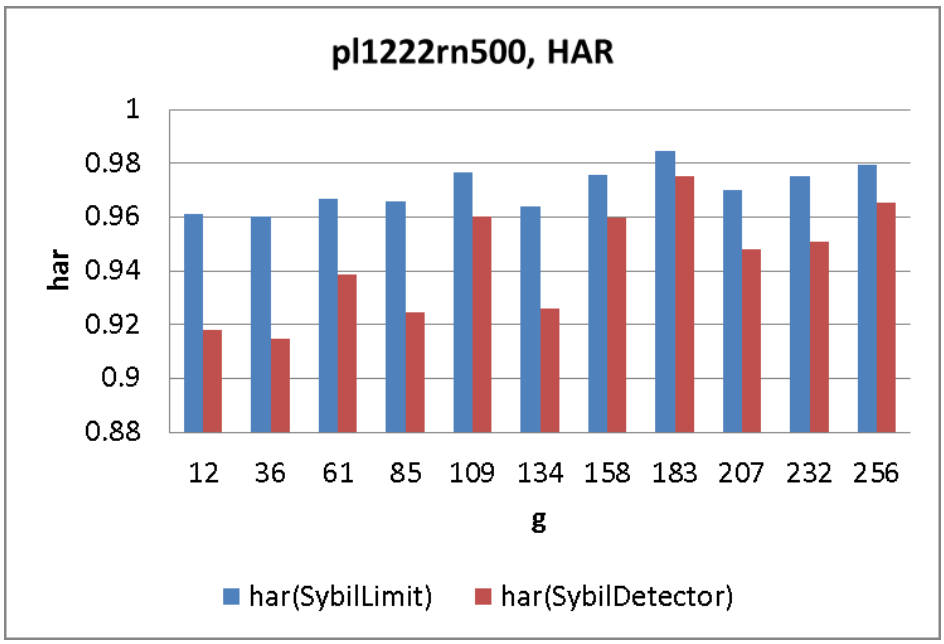


(b) G2

Figure 3.6: Changes of sar as snn increases

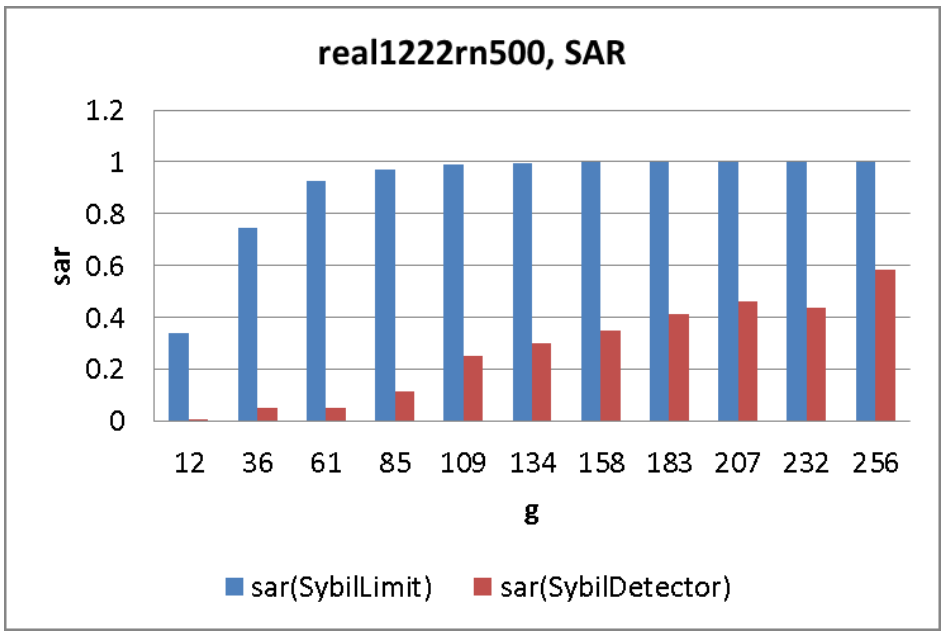


(a) G1

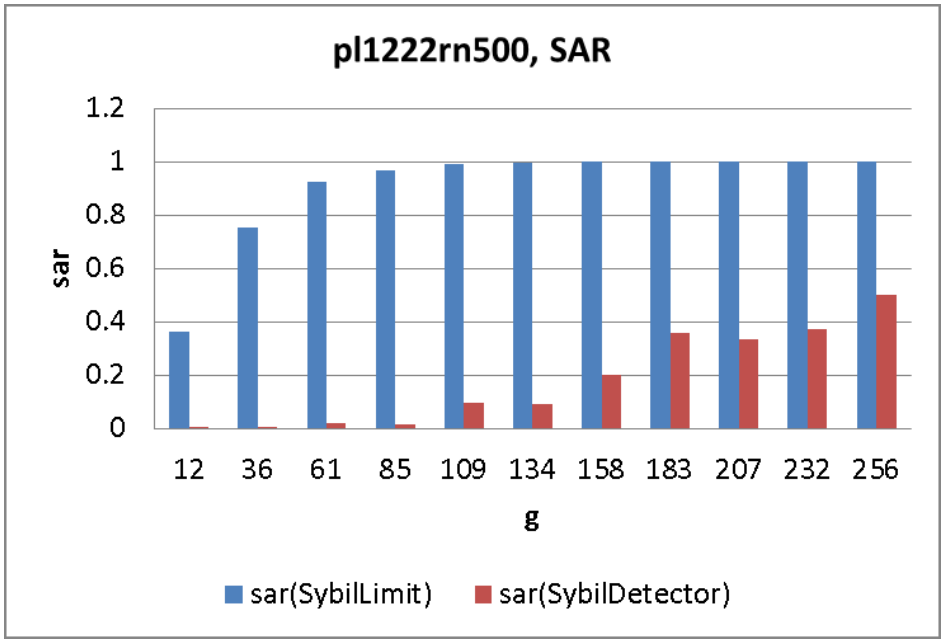


(b) G2

Figure 3.7: Changes of har as g increases



(a) G1



(b) G2

Figure 3.8: Changes of sar as g increases

and the synthetic network topologies, when the g is 109, the sar of SybilLimit increases to 98%. Meanwhile, the sar of SybilDetector is 4x and 10x lower in the real world network topology and the synthetic network topology, respectively.

In summary, this section compared the performance of SybilDetector and SybilLimit by simulation on real and synthetic network topologies. It is shown that the accuracy of SybilDetector is significantly higher than that of SybilLimit. Additionally, the potential of attack edge detecting in creating effective Sybil resisting algorithms is confirmed.

3.5 Discussion

One potential problem of SybilDetector is that it assumes that the network has only two clusters, the honest region and the Sybil region. It is reasonable to consider that honest nodes are also divided into multiple clusters in real world systems. Indeed, this is a common problem for all existing SSD algorithms. Viswanath et al.[52] has highlighted that, in a DCS, the accuracies of existing SSD algorithms decrease if the honest region contains cluster structures. To detect Sybil nodes in DCSs where the honest region contains multiple clusters, honest nodes need more knowledge of the systems. For example, for each honest node v and each honest cluster c, suppose that v knows the type of at least one node u in c. Using existing SSD algorithms, u can detect the honest nodes that belong to c, and report such information to v. In this way, v can distinguish honest nodes from Sybil nodes. Designing Sybil resisting mechanisms for DCSs with more than two clusters will be part of the future work of this dissertation.

3.6 Conclusion

The Sybil attack is a critical security threat to DCSs. SSD algorithms are the representative solutions to this attack. However, existing SSD algorithms are inaccurate. Thus, the objective of this chapter is to design more accurate SSD algorithms.

This chapter proposed SybilDetector, an accurate SSD algorithm. To obtain high accuracy, the core innovation of SybilDetector is to detect the attack edges. The accuracy of SybilDetector was compared with that of SybilLimit – a representative existing SSD algorithm, on both synthetic and real world social network topologies. In the simulations, the sar of SybilDetector was at least 4x lower than that of SybilLimit.

Moreover, the accuracy improvement made by SybilDetector clearly confirmed that attack edge detecting is a promising technique to resist the Sybil attack. It is expected that this technique can be used to create effective security mechanisms for other DCS attacks.

Chapter 4

RSC: an Attack Edge Detecting Algorithm for Sybil Resisting

4.1 Introduction

The objective of this chapter is to design an attack edge detecting algorithm for unauthorized DCSs – DCSs that do not contain trustful authorities. Designing such an algorithm will allow accurate SSD algorithms for unauthorized DCSs to be created.

Unauthorized DCSs form an important family of DCSs, and attack edge detecting-based SSD algorithms are needed to protect them. Unauthorized DCSs are more scalable than authorized DCSs. In an authorized DCS, the trustful authorities have to provide resources to other nodes. Accordingly, the capacities of trustful authorities limit the size of the system. Moreover, in many DCSs such as ad hoc systems, maintaining trustful authorities is expensive[53]. Therefore, many existing SSD algorithms such as SybilLimit[18] and Gatekeeper[17] have been designed for unauthorized DSCs, and it is necessary to design attack edge

detecting-based SSD algorithms for this type of DCS.

As discussed in Section 3.2.5, the basic steps to create an attack edge detecting algorithm are as follows. First, design a (or select an existing) detecting metric. Here, edge betweenness metrics measure the extent to which edges are passed by message paths. A detecting metric is an edge betweenness metric that satisfies the detecting property – the betweennesses of the attack edges are higher than those of the non-attack edges. Then, design a mechanism that enables each node to securely compute the betweennesses of edges in a distributed manner.

To create attack edge detecting algorithms for unauthorized DCSs, the main difficulty is to design the appropriate detecting metric. This metric should satisfy the detecting property, and it should be possible to securely compute values of this metric in unauthorized DCSs. So far, the only edge betweenness metric that is known to satisfy the detecting property is the SPEB (shortest path edge betweenness). However, the SPEB-based attack edge detecting algorithms can only be implemented in authorized DCSs as described in Section 3.2.5.

This chapter makes the following contributions.

1. It analyzes the properties of the random walk edge betweenness (RWEB, Section 3.2.5) and presumes that the RWEB is an appropriate detecting metric for unauthorized DCSs. Evaluation on synthetic and real world network topologies finally validates this presumption.

2. It designs an algorithm for unauthorized DCSs, called Random walk and SNM-based Clustering (RSC), enabling each honest node to distinguish the attack edges among its incident edges. Basically, RSC enables each node to compute RWEBs of its incident edges in a distributed manner. Since the RWEB satisfies the detecting property, nodes can distinguish attack edges

among their incident edges. The difficulty in the design of RSC is to resist attacks from Sybil nodes.

3. It provides an example showing how RSC can be used to create more accurate SSD algorithms. RSC is incorporated into SOHL[19] – an existing unauthorized SSD algorithm, to create an unauthorized SSD algorithm called *RSC-based Sybil resisting* (RSSR). Evaluation shows that RSSR makes a remarkable accuracy improvement over SOHL.

This chapter is organized as follows. Section 4.2 introduces the related research. Sections 4.3 elaborates the design of RSC. Then, Section 4.4 integrates RSC into SOHL to create RSSR. Section 4.5 evaluates the performances of RSC and RSSR. Finally, Section 4.6 concludes this chapter. *

*This chapter is an extended version of [54].

4.2 Related Work

4.2.1 Random Walk-based DCSs

In many DCSs, nodes communicate using random walks[55, 56, 57, 58]. For example, Lv et al.[59] showed that random walks can be used to create efficient searching algorithms in P2P systems. Also, as shown in Section 3.2.3, the random walk is an essential constructing component for SSD algorithms. Random walk-based DCSs are usually strong against network change: information is spread among the systems probabilistically so that nodes can keep communicating during network change.

4.2.2 Random Walk Betweenness

Section 3.2.4 briefly explained the RWNB (random walk node betweenness) and the RWEB, two betweenness metrics defined by Newman[41]. The RWEB is to be used to detect attack edges in this chapter. Therefore, this section introduces the formal definitions of these two metrics.

In a DCS, for each pair of nodes s and t, s and t disseminate one *absorbing random walk* (ARW) to each other. Here, an absorbing random walk from s to t, denoted by (s,t)-ARW, is a message packet: starting from s, this packet advances in a random walk manner continuously until it reaches t. Let arw be an average (s,t)-ARW. For each edge, $e=(v,u)$, the expected numbers of times that arw passes e from v to u (from u to v) is denoted by $e.i_v^{(s,t)}$ ($e.i_u^{(s,t)}$). Then,

$$e.i^{(s,t)} = |e.i_v^{(s,t)} - e.i_u^{(s,t)}| \tag{4.1}$$

is called the *partial betweenness of* e *for the* (s,t) *random walks*. Namely, $e.i^{(s,t)}$

represents the "pure" expected number of times that a (s,t)-ARW passes e. Now, the RWEB of e is defined as

$$e.i = \sum_{\forall(s,t)} e.i^{(s,t)}. \tag{4.2}$$

Let $E(v)$ and $N(v)$ denote the incident edges and incident nodes of v, respectively. The summation of the betweennesses of the incident edges of v, namely

$$\sum_{\forall e=(v,u)\in E(v)} e.i, \tag{4.3}$$

is defined as the RWNB of v. The good property of the RWNB is that, in a DCS, the RWNBs of *front nodes* are higher than those of other nodes. Here, front nodes are the nodes connecting multiple clusters.

4.3 RSC – Detecting Attack Edges

This section introduces RSC, an algorithm that enables each node to distinguish the attack edges among its incident edges in unauthorized DCSs. RSC is designed in two steps. The first is to choose an edge betweenness metric as the detecting metric. Under this metric, RSC enables each node to securely compute the betweennesses of its incident edges. Given this betweenness knowledge, each node can distinguish the attack edges among its incident edges. Section 4.3.1 explains the reason why the RWEB is a potential detecting metric for unauthorized DCSs. Section 4.3.2 then introduces the design detail of RSC.

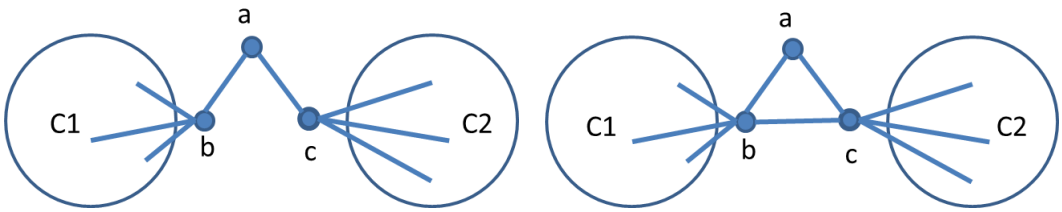
4.3.1 Choice of Detecting Metric

A detecting metric for unauthorized DCS needs to satisfy two properties: 1). this metric satisfies the detecting property; 2). it is easy to securely compute values of this metric in unauthorized DCSs.

This dissertation presumes that the RWEB satisfies the detecting property, based on the following observations.

1. The RWNB of each node v is the summation of the RWEBs of incident edges of v. When v is a front node, v has a high RWNB no matter how many incident edges v has. The attack edges are edges incident to front nodes. Hence, the incident edges of v, including the attack edges incident to v, should on average have high RWEBs.

2. An ARW from an honest node to a Sybil node is called a *first-type ARW*. Similarly, an ARW from an honest node to another honest node is called a *second-type ARW*. Since 1). all first-type ARWs have to pass the attack edges, 2). the number of attack edges is small, and 3). the number of Sybil



(a) before node b and c are connected (b) after node b and c are connected

Figure 4.1: Influence of network change on shortest paths and random walks

nodes is large, it is also reasonable to presume that the attack edges have high RWEBs.

Moreover, it is harder for Sybil nodes to interrupt the computing of the RWEB than to interrupt the computing of the SPEB, which makes it possible to securely compute the RWEB in unauthorized DCSs. The basic approach for Sybil nodes to interrupt the computing of betweennesses of edges is to change the network topology. In a DCS, a slight change of the network topology may significantly change all the shortest paths between nodes. In contrast, the influence of network change on random walks is smaller.

Figure 4.1 shows an example of the influence of the network change on shortest paths and random walks (by Newman[41]). In this figure, before b and c are connected, all shortest paths and random walks between clusters $C1$ and $C2$ have to pass node a. When nodes b and c are connected (network changes), all shortest paths between $C1$ and $C2$ change, where no shortest paths pass a any more. In contrast, when nodes b and c are connected, only some of the random walks change, where some random walks still pass a.

Therefore, it is harder for Sybil nodes to interrupt the computing of the RWEB, making it possible to compute the RWEB without trustful authorities.

Based on the above consideration, this dissertation regards the RWEB as a candidate betweenness metric for detecting attack edges in unauthorized DCSs.

4.3.2 Distributed Computing of the RWEB

This section presents RSC. Henceforth, the RWEB is simply called the betweenness for short. In the following section, a basic protocol of RSC is first designed. Then, potential attacks against this basic RSC and mechanisms resisting these attacks are discussed.

The number of honest nodes in the system is denoted by n. The edges among honest nodes are called the honest edges, and the number of honest edges is M. The number of attack edges in the system is denoted by g. It is assumed that each node knows n and M, consistent with existing SSD algorithms[17, 19, 20]. Table 4.1 lists the important denotations used in this chapter.

Basic RSC Protocol

Each node v has a unique ID, which is simply the hash value of the IP address of v. Additionally, v maintains a *destination set*, denoted by $v.Des$, that stores the IDs of other nodes in the system. For each node u in $v.Des$, v disseminates $v.ir$ (v,u)-ARWs. Here, $v.ir$, called the *issue rate* of v, is a local parameter set by v itself. In practice, each node sets its issue rate to be as large as possible according to its computing capacity. RSC does not assume that the destination set of v contains IDs of all the nodes in the system. Instead, once a node u requires node v for communication, v adds u to $v.Des$. For each incident edge $e = (v,u)$ and each set of (s,t)-ARWs, v holds two variables $v.u_{-}^{(s,t)}$ and $v.u_{+}^{(s,t)}$. When a (s,t)-ARW rw enters (leaves) v from e, v increases $v.u_{-}^{(s,t)}$ ($v.u_{+}^{(s,t)}$) by one. In this way, $v.u_{-}^{(s,t)}$ and $v.u_{+}^{(s,t)}$ are approximations of $e.i_{v}^{(s,t)}$ and $e.i_{u}^{(s,t)}$, respectively. Then, v can compute the partial betweenness of e for the (s,t)-ARWs, denoted by $v.i_{u}^{(s,t)}$, as

Table 4.1: Important denotations

Name	Description
ARW	Absorbing random walk
(s,t)-ARWs	The absorbing random walks between node s and node t
First-type ARWs	The ARWs between nodes of different types
Second-type ARWs	The ARWs between honest nodes
n	The number of honest nodes in the system
Honest edges	The edges among honest nodes
M	The number of honest edges in the system
har	The probability that node v accepts node u, where v and u are two random honest nodes
sar	The probability that node v accepts node u, where v and u are a random honest node and a random Sybil node, respectively
er	The probability that a random walk starting from the honest region enters the Sybil region
$N(v)$	The incident nodes of node v
$E(v)$	The incident edges of node v
RWNB	The random walk node betweenness
RWEB	The random walk edge betweenness

$$v.i_u^{(s,t)} = |v.u_-^{(s,t)} - v.u_+^{(s,t)}|. \tag{4.4}$$

Finally, the betweenness of edge e is computed as

$$v.i_u = \sum_{\forall(s,t)} v.i_u^{(s,t)}. \tag{4.5}$$

Resisting Attacks

This section shows how Sybil nodes can break the protocol of the basic RSC and provides solutions to these attacks. More general attacks such as DoS and message forging[60] are out of the discussion scope of this dissertation.

Attack 1. *Let arw be a second-type (s,t)-ARW. Once arw enters the Sybil region, Sybil nodes discard arw.*

Attack 2. *Let arw be a first-type (s,t)-ARW. Sybil nodes reduce the betweennesses of attack edges by manipulating the path of arw. Let $e = (v,u)$ be an attack edge, where v and u are honest and Sybil, respectively. Suppose that u has received arw from v, u always sends arw back to v. Accordingly, the betweenness of e will always be zero. An example of this attack is shown in Figure 4.2: nodes v and u are connected by edge e, where v is honest and u is Sybil. To reduce the betweenness of e, on receiving a random walk packet from v, u simply returns this random walk back to v.*

To resist Attacks 1 and 2, the following lemma is needed.

Lemma 1. *In a DCS with no attack, each first-type ARW increases the betweenness of at least one attack edge.*

Proof. Let rw be a first-type ARW, and let k denote the number of times that rw traverses attack edges during its movement. In a secure environment, rw

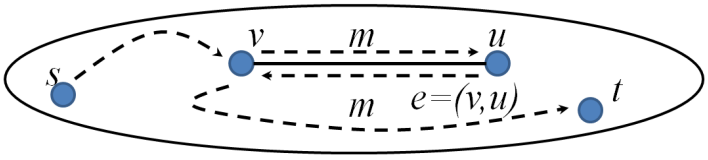


Figure 4.2: Attack 2

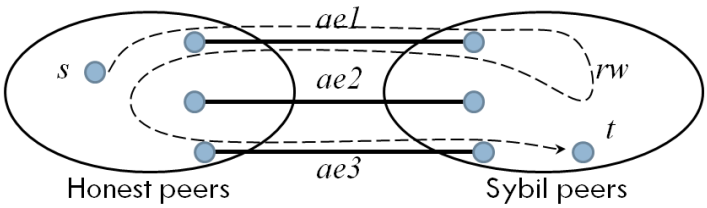


Figure 4.3: First-type ARWs pass attack edges odd times

will finally stop in the Sybil region. Therefore, k is an odd number. Figure 4.3 provides an example showing this property. The dot dash line is a first-type (s,t)-ARW, denoted by rw. $ae1$, $ae2$ and $ae3$ are the attack edges of the system. The number of times that rw traverses the attack edges is an odd number of three. Supposing that the first $k-1$ times of traverses cancel each other out completely, the left traverse will still pass a certain attack edge from the honest region to the Sybil region. Therefore, the betweenness of at least one attack edge will be increased. □

RSC uses a *Distance Limitation* (DL) technique to resist Attacks 1 and 2: for each (s,t)-ARW, arw, once arw has hopped $\Theta(M)$ steps, arw stops and s rejects t. To implement DL, arw contains a counter initiated to be $\Theta(M)$. On each hop, the node u that arw currently lies on decreases the counter of arw by one. If the counter is zero, u discards arw and asks s to reject t.

The following showes that, under DL, 1). Sybil nodes do not launch Attack 1 or Attack 2, 2). the probability that honest nodes reject each other is low, and,

3). attack edges have high betweennesses.

Lemma 2. *Under DL, Sybil nodes do not launch Attack 1 or Attack 2.*

Proof. Under Attack 1, t will be rejected by s. However, arw now turns into a first-type ARW, which increases the betweenness of at least one attack edge. Indeed, if Attack 1 is launched, s can significantly increase the betweennesses of attack edges by sending many (s,t)-ARWs. Hence, Sybil nodes should not launch Attack 1.

As for Attack 2, since t is selected from the destination set of s, Sybil nodes will expect t to be accepted by s. Therefore, Sybil nodes should not launch Attack 2. Indeed, to prevent t from being rejected by s, once arw enters the Sybil region, Sybil nodes should send arw to t directly. □

Lemma 3. *Under DL, the probability that honest nodes reject each other is low.*

Proof. Let arw be a second-type (s,t)-ARW. It is sufficient to show that, having moved $\Theta(M)$ steps, the probability that arw does not reach t is low. As introduced in Section 3.2.3, after $O(\log n)$ steps, arw traverses each edge with a probability of $\Theta(\frac{1}{M})$ on each step. Since $\Theta(M)$ is much larger than $O(\log n)$, it can be considered that arw traverses each edge with an equal probability $\Theta(\frac{1}{M})$ since the first step. For node t, call the number of the incident edges of t the *degree* of t, denoted by $k(t)$. The probability that arw does not arrive at t in $\Theta(M)$ steps is

$$\begin{aligned} & (1-\Theta(k(t)/M))^{\Theta(M)} \\ < \ & (1-\Theta(1/M))^{\Theta(M)} \\ = \ & e^{-1}. \end{aligned} \tag{4.6}$$

This means that, as M increases, the probability that arw does not arrive at t in $\Theta(M)$ steps is smaller than a constant $1/e$. Lemma 3 follows. □

Lemma 4. *Under DL, attack edges have high betweennesses.*

Proof. Let arw be a first-type (s,t)-ARW. According to Lemma 1, it is sufficient to show that arw reaches Sybil node t in $\Theta(M)$ steps with a high probability. This probability is

$$\begin{aligned} & 1-(1-\Theta(g/M))^{\Theta(M)} \\ > \; & 1-(1-\Theta(1/M))^{\Theta(M)} \\ = \; & 1-e^{-1}. \end{aligned} \tag{4.7}$$

This means that as M increases, the probability that arw reaches its destination is larger than a constant $1-1/e$. Lemma 4 follows. □

Therefore, DL can protect RSC to securely compute the betweennesses of edges.

4.3.3 Distinguish Attack Edges

Having computed the betweennesses of incident edges, each node v can judge the possibility that a certain incident edge is an attack edge. Specifically, v computes the *relative suspect rate* of e as

$$v.sus(e) = \frac{v.i_u}{\sum_{\forall e=(v,u)\in E(v)} v.i_u}.$$

Section 4.4 shows how the suspect rate can be used to create accurate SSD algorithms.

4.4 RSSR

This section provides an example to clarify the potential of RSC in improving the accuracy of existing SSD algorithms. Specifically, this section reduces the sar of SOHL (briefly introduced in Section 3.2.3) using RSC. The resulting algorithm is called RSSR.

4.4.1 Sybil Resisting One Hop Lookup

First, an introduction of SOHL is given below. In SOHL, each honest node disseminates r probing random walks. As introduced in Section 3.2.3, a probing random walk is a message that moves $O(\log n)$ steps in a random walk manner. The ending nodes of the probing random walks of v consist of the *finger set* of hn ($hn.fingers$). Finally, hn accepts node u if and only if u is in $hn.finger$, or in finger sets of the nodes in $hn.finger$. Since the probing random walk has a low escape rate, the sar of SOHL is low.

4.4.2 Incorporating RSC into SOHL

The basic protocol of RSSR is the same as that of SOHL, except for one difference: in RSSR, the nodes disseminate *waterfall random walks* instead of normal probing random walks. Let rw be a random walk of a length $\log n$. Suppose that rw is currently on node v, and $e = (v, u)$ is an incident edge of v. The probability that v selects u as the next hop for rw is inversely proportional to $v.sus(e)$. Such a random walk is called a waterfall random walk. Since the attack edges are expected to have high betweennesses, the escape rate of the waterfall random walk should be lower than the escape rate of the probing random walk. Hence, the sar of RSSR should be lower than that of SOHL.

4.5 Evaluation

This section has two goals. The first is to evaluate the performance of RSC by simulation. RSC is expected to generate betweennesses that satisfy the detecting property. Hence, the performance of RSC is measured by the ratio between the average betweenness of attack edges and the average betweenness of honest edges. To this end, simulations are implemented in the following manner. In each simulation, each node v randomly chooses another node u from the system, and disseminates air (v, u)-ARWs (air means *ARW issue rate*). The average betweenness of honest edges is defined as the *honest edge betweenness*

$$heb = average(\{\frac{v.i_u + u.i_v}{2} \mid \text{for each honest edge } e = (v, u)\})/air,$$

and the average betweenness of attack edges is defined as the *attack edge betweenness*

$$aeb = average(\{\frac{v.i_u + u.i_v}{2} \mid \text{for each attack edge } e = (v, u)\})/air.$$

Then, the *detecting factor* $df = aeb/heb$ is the metric for the performance of RSC. Each simulation is run five times, and the results are averaged.

The second goal is to validate the performance improvement made by RSSR over SOHL by simulation. As RSSR and SOHL are both SSD algorithms, the performances of RSSR and SOHL are evaluated by har and sar. During each simulation, the length of a probing random walk (waterfall random walk) is $\log n$, and the number of probing random walks (waterfall random walks) disseminated per node is $\sqrt{M}$. Each simulation is run five times, and the results are averaged.

4.5.1 Network Construction

Each network used in the simulations is created as follows. First, the networks of the honest region and Sybil region are created separately. The two regions are then connected by g attack edges. Three networks are used to create the honest region. One is a real world social network that represents the hyperlinks among the blogs created during the 2005 U.S. election[48]. This network is used because it represents a complete social network. The other two networks are synthetic networks generated according to the Barabasi-Albert (BA) model[49] using NetworkX[50]. The BA model is designed to present real world social networks. The networks of the Sybil regions are created using the Erdos-Renyi (ER) model[51]. In a network obeying the ER model, each pair of nodes are connected by an edge with a probability of p. It is impossible to presume the network topology of a Sybil region because Sybil nodes can randomly change the connection between each other. Hence, this evaluation simply uses the ER model to create the Sybil regions. In this evaluation, p is set to 0.05. Table 4.2 lists the details of networks used to create the honest and Sybil regions. Using these networks, three networks G1, G2 and G3 are created for simulations, as listed in Table 4.3.

4.5.2 Simulation Results

Influence of air

Figure 4.4 shows the influence of air on the performance of RSC. G3 is used for this evaluation, and g is 30. Three points are notable. First, generally, both aeb and heb gradually decrease as air increases. By definition, the betweenness of e is the expectation of the pure number of times that ARWs pass e. As introduced

Table 4.2: Networks used for creating honest and Sybil regions

Name	Type	Number of nodes	Number of edges
real1222	Real world social network	1,222	16,714
pl1222	Synthetic network of Barabasi-Albert model	1,222	7,257
rn500	Synthetic network of Erdos-Renyi model	500	1,725
pl100	Synthetic network of Barabasi-Albert model	100	545
rn100	Synthetic network of Erdos-Renyi model	100	149

Table 4.3: Networks used for evaluation

Network name	Honest region	Sybil region
G1	real1222	rn500
G2	pl1222	rn500
G3	pl100	rn100

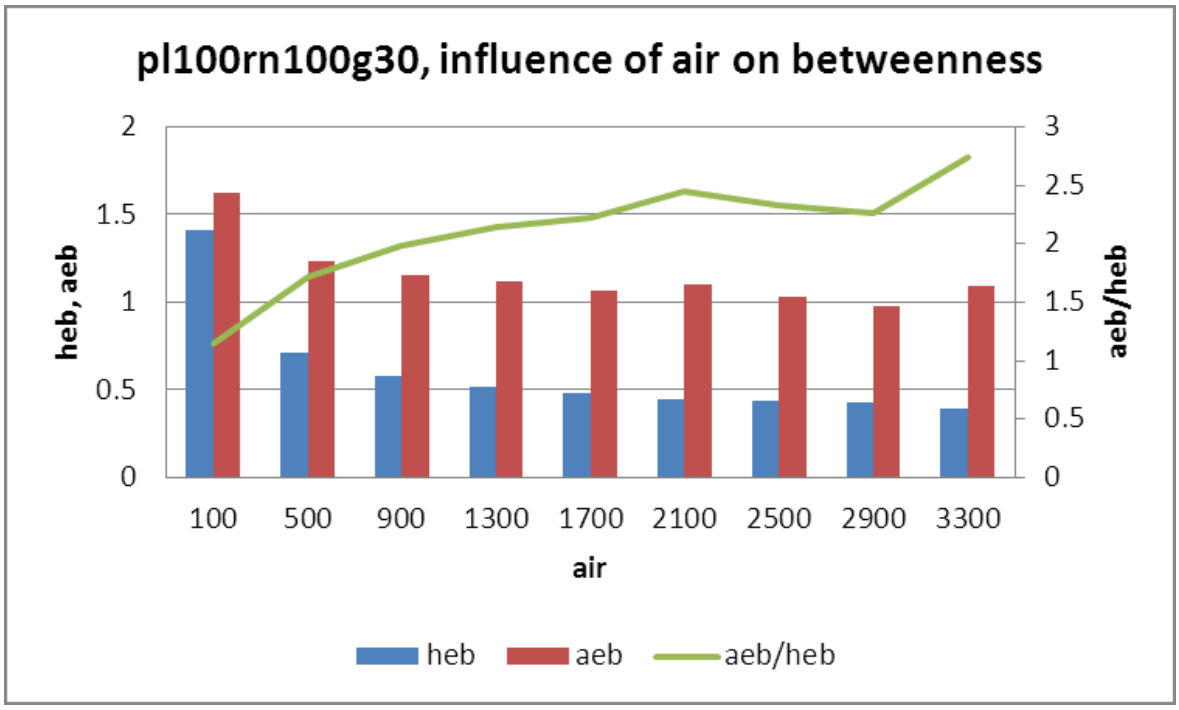


Figure 4.4: Influence of air on the betweenness

in Section 3.2.3, in a SNM-based DCS, random walks starting from the honest region mix in a short distance. When air is large, many ARWs disseminated by nodes mix before reaching their respective destinations. Suppose that arw is such a mixed ARW. On each step, arw passes each edge e from the opposite directions with the equal probability of $1/(2M)$, decreasing the betweenness of e. Therefore, as air increases, aeb and heb decrease.

Second, as air increases, aeb and heb finally become stable. Originally, the edge betweenness is a probabilistic definition. Hence, for a certain network, the betweenness distribution of this network is fixed. RSC computes the betweennesses of edges in a Monte Carlo manner. As air increases, the betweennesses of edges computed by RSC should approach the true distribution of the betweenness. Therefore, aeb and heb should finally become stable.

Third, as aeb and heb become stable, aeb is higher than heb. According to Lemma 1, this is attributed to the dissemination of first-type ARWs.

The above results indicate that, to ensure the performance of RSC, air should be sufficiently large. In practice, this can be ensured by running RSC periodically.

Influence of first-type ARWs

Figure 4.5 validates that the performance of RSC is closely relevant to the dissemination of first-type ARWs. G3 is used for this evaluation, with g being set to 30. Lemma 1 claims that the betweennesses of attack edges increase if more first-type ARWs are disseminated. In Figure 4.5, the blue, red and green bars represent the df obtained when 1). both first-type ARWs and second-type ARWs are disseminated, 2). only first-type ARWs are disseminated, and 3). only

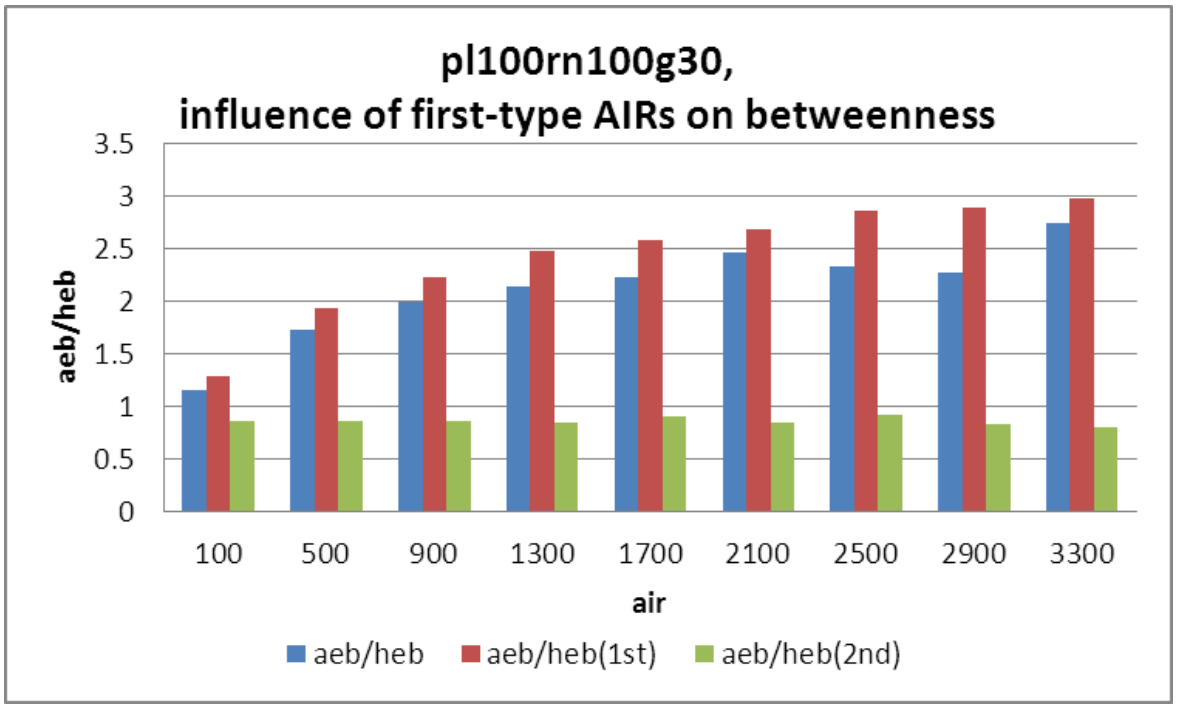


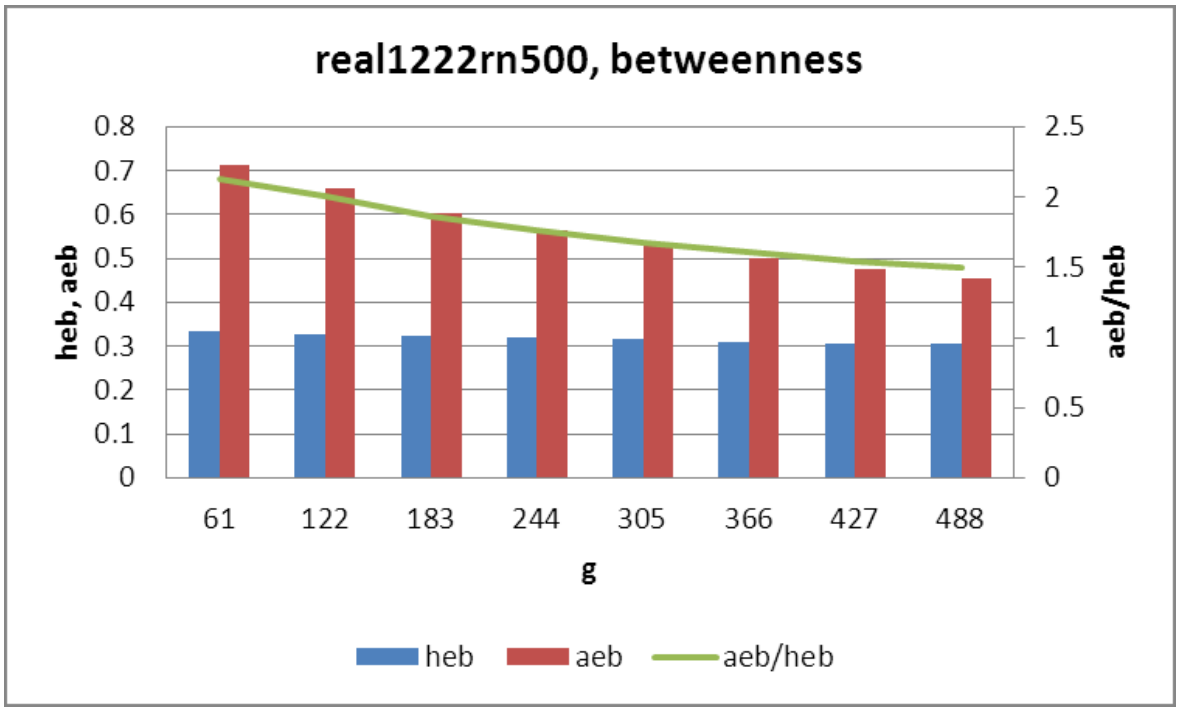
Figure 4.5: Influence of first-type ARWs on df

second-type ARWs are disseminated, respectively. Clearly, the df of RSC increases when more first-type ARWs are disseminated, validating Lemma 1.

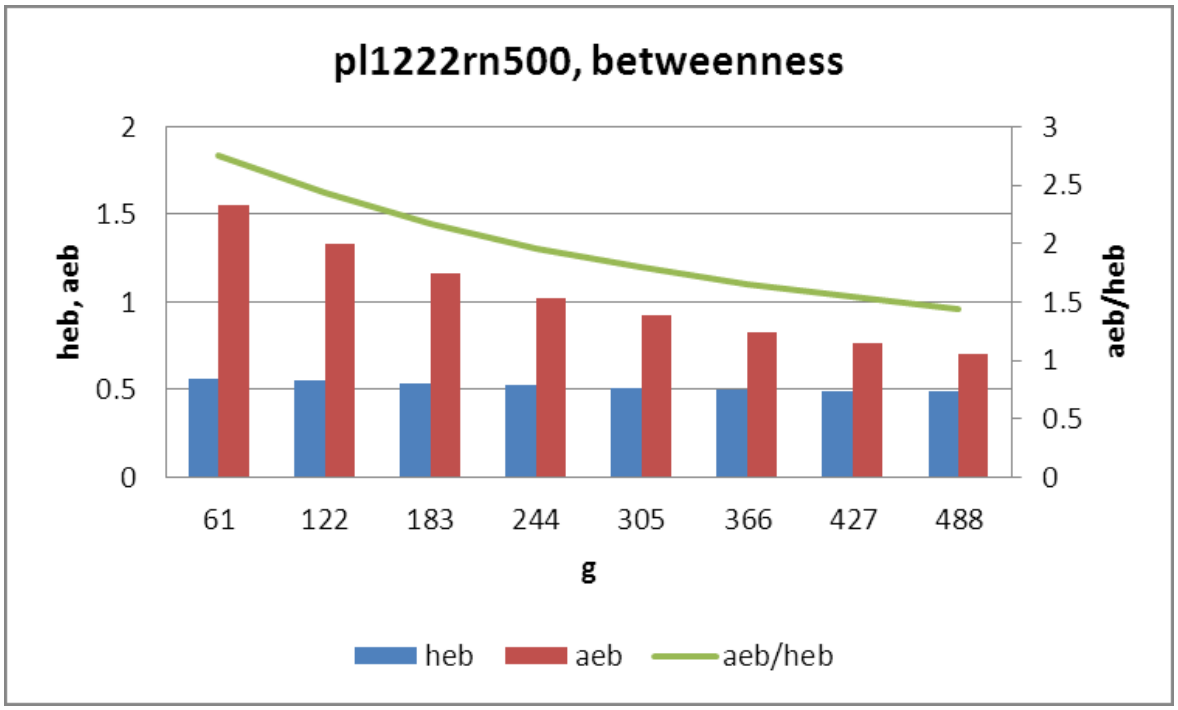
Influence of g

In the following evaluations, air is set to 8000. Other values of air ranging from 1500 to 8000 have also been evaluated. Generally, as air increases, the performances of RSC and RSSR increase accordingly.

Figure 4.6 shows the influence of g on the performance of RSC. In this evaluation, networks G1 and G2 are used. It is notable to see that aeb decreases as g increases. Intuitively, as g increases, the average number of first-type ARWs passing each attack edge decreases, and hence aeb decreases. However, heb is lower than aeb in both the real world and synthetic network topologies. Therefore, RSC can be regarded as a feasible attack edge detecting algorithm.

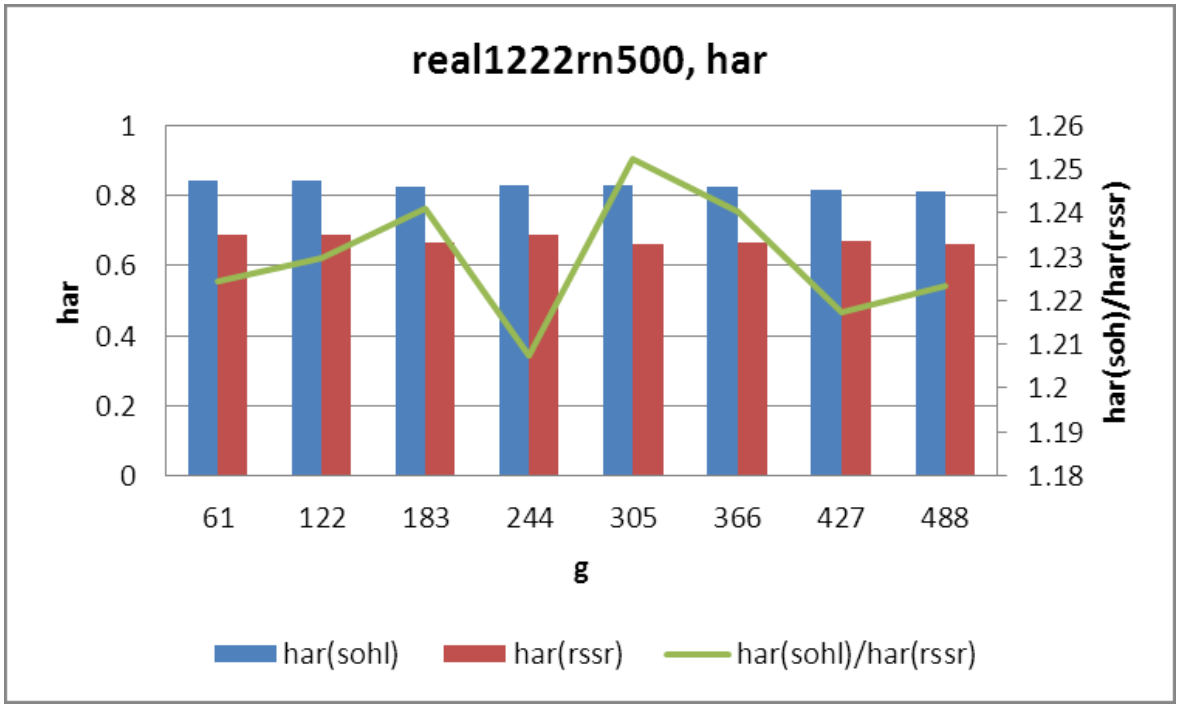


(a) G1

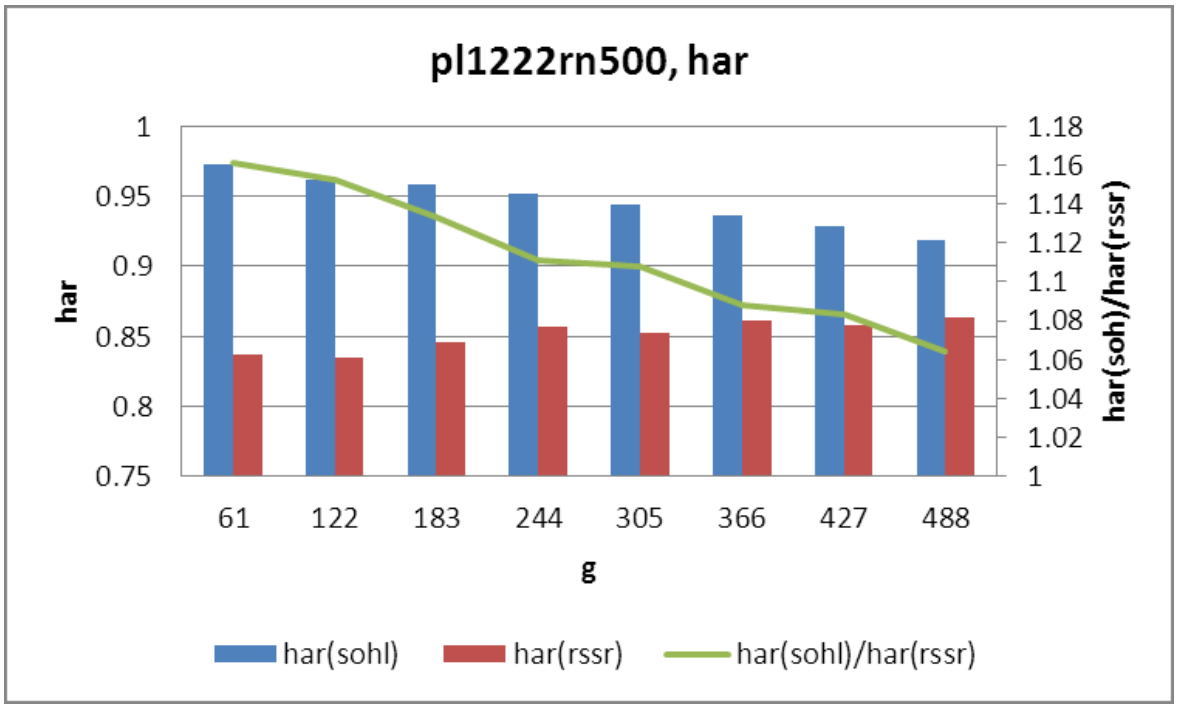


(b) G2

Figure 4.6: Influence of g on heb and aeb



(a) G1



(b) G2

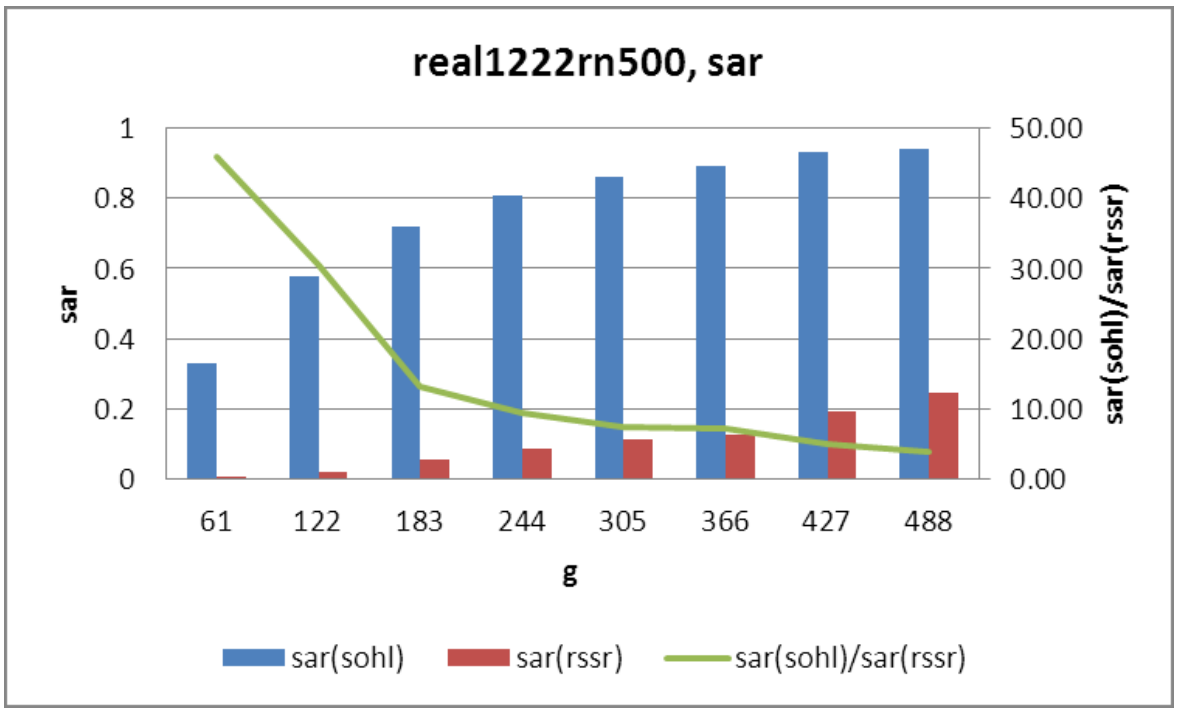
Figure 4.7: Influence of g on har

Figure 4.7 shows the influence of g on the har of SOHL and RSSR. Networks G1 and G2 are used for this evaluation. In both the real world and synthetic network topologies, the har of RSSR is 20% lower than the har of SOHL. In RSSR, RSC can detect the attack edges and prevent waterfall random walks from approaching the attack edges. Accordingly, honest nodes near the attack edges are less likely to be visited by waterfall random walks and are therefore less likely to be accepted by other honest nodes. Hence, the har of RSSR is lower than that of SOHL.

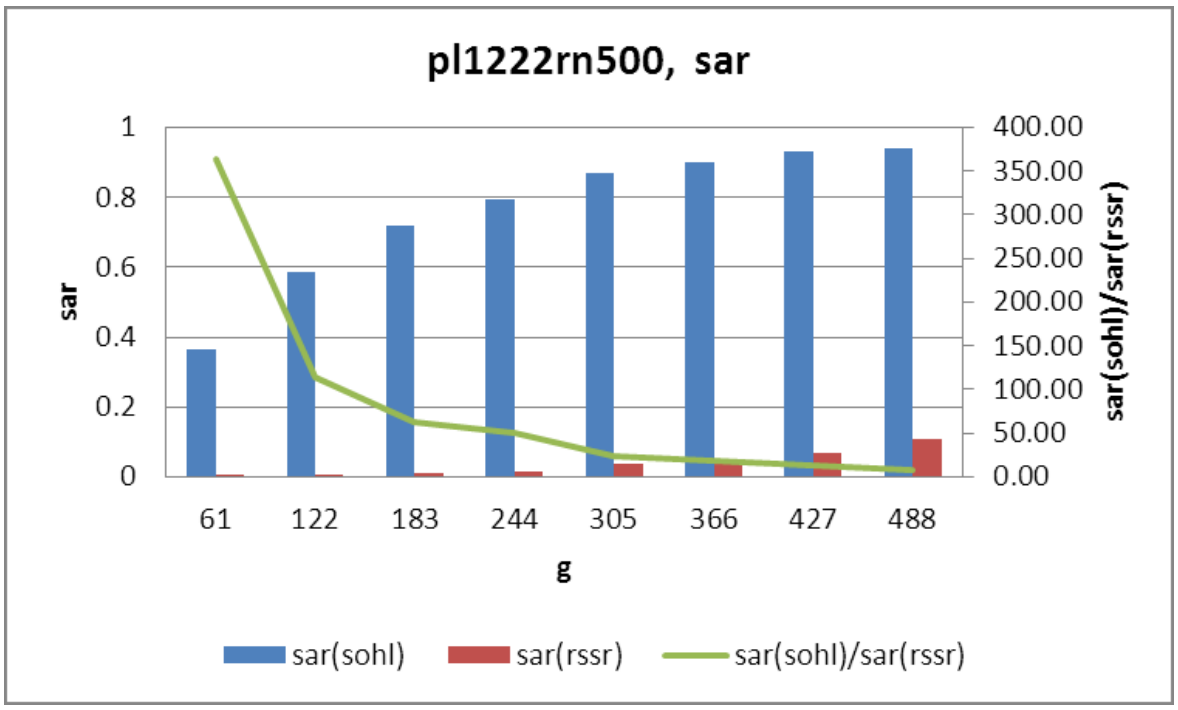
Note that RSSR is only designed to have low sar, instead of having high har. The results above also indirectly indicate that RSC can effectively detect the attack edges.

Figure 4.8 shows the influence of g on the sar of SOHL and RSSR. Networks G1 and G2 are used for this evaluation. First, as g increases, the sar of both RSSR and SOHL increase. As g increases, aeb decreases, and the escape rate of waterfall random walks increases. Accordingly, honest nodes accept more Sybil nodes, increasing the sar. Second, the sar of RSSR is lower than that of SOHL. For example, when $g = 427$, the sar of SOHL increases to 90%. However, the sar of RSSR is 5x and 13x times lower than that of SOHL using real world and synthetic network topologies, respectively. These results validate that RSC can remarkably improve the accuracy of SOHL.

In this section, simulations on synthetic and real world network topologies show that the betweenness values computed by RSC satisfy the detecting property. Hence, RSC is a feasible attack edge detecting algorithm for unauthorized DCSs. Moreover, it has also shown that, by utilizing RSC, the accuracy of SOHL is significantly improved. Therefore, these results validate the power of RSC in creating accurate SSD algorithms.



(a) G1



(b) G2

Figure 4.8: Influence of g on sar

4.6 Conclusion

The goal of this chapter is to design an attack edge detecting algorithm for unauthorized DCSs. Having such an algorithm, accurate SSD algorithms for unauthorized DCSs can be created based on attack edge detecting.

To this end, this chapter proposed RSC, an algorithm that enables each node to distinguish the possible attack edges among its incident edges. The main difficulty in the creating of RSC is to find a detecting metric that can be securely computed in unauthorized DCSs. By analyzing the properties of existing betweenness metrics, this dissertation chooses the RWEB as such a metric. RSC enables each node to securely compute the RWEBs of its incident edges in a distributed manner. The difficulty is to resist attacks from Sybil nodes. Simulations on real world and synthetic network topologies showed that the betweenness computed by RSC satisfies the detecting property well: the betweennesses of attack edges are higher than those of honest edges. This validates the feasibility of RSC.

This chapter also provided an example showing that RSC can be readily used to improve the accuracies of existing SSD algorithms. That is, RSC is embedded into SOHL to create a new SSD algorithm. The evaluation showed that RSC remarkably reduced the sar of SOHL.

In summary, this chapter created a feasible attack edge detecting algorithm for unauthorized DCSs, which can be used to create accurate SSD algorithms for unauthorized DCSs.

Chapter 5

Conclusions

DCSs, such as P2P systems, ad hoc network systems and volunteer computing systems, are playing pivotal roles in industry and our daily life. However, malicious nodes may exist in DCSs and launch attacks against the systems. Especially, the false result attack and the Sybil attack are two representative attacks to DCSs. For a DCS, under the false result attack, malicious worker nodes deliberately send incorrect results of computing tasks to host nodes. Under the Sybil attack, malicious users control many Sybil nodes to attack the system. To ensure the application and development of DCSs, it is necessary to resist these two attacks. However, existing solutions to these two attacks are problematic. Hence, this dissertation aims to design more effective attack resisting mechanisms.

Chapter 2 proposed a false result attack resisting algorithm MSC, which enables hosts to detect malicious workers efficiently without using quizzes. The core innovation of MSC is to detect malicious workers using normal tasks called checking tasks, instead of quizzes. In MSC, for each host and its workers, the host disseminates checking tasks to each worker. The host then judges each worker by checking whether the worker returned correct results to the checking

tasks. Since malicious workers return incorrectAdditionally results, over time, the host can detect malicious workers. Theoretical analysis and simulations showed that each host can accurately detect malicious workers in reasonable DCSs. In addition, the efficiency of MSC is theoretically optimal. Hence, MSC is a more efficient and practical solution to the false result attack compared to existing solutions.

Chapter 3 proposed SybilDetector, an accurate SSD algorithm for authorized DCSs. This dissertation observed that, to increase the accuracy of SSD algorithms, it is necessary to further prohibit the communication between nodes of different types. To this end, the intuitive idea is to directly detect and cut the attack edges. In SybilDetector, honest nodes first detect the attack edges using a SPEB-based mechanism. Then, two nodes accept each other if and only if the shortest paths between these two nodes do not pass any attack edges. Evaluation showed that SybilDetector made a significant improvement in accuracy over an existing representative SSD algorithm. Moreover, Chapter 3 confirmed that attack edge detecting is an important technique in resisting the Sybil attack. It is expected that this technique can be widely used to create security mechanisms for other DCS attacks in the future.

Chapter 4 proposed an attack edge detecting algorithm for unauthorized DCSs called RSC, which can be used to create accurate SSD algorithms for unauthorized DCSs. As shown in Chapter 3, it is crucial to detect attack edges in order to create accurate SSD algorithms. However, how to detect attack edges in unauthorized DCSs is an open problem. To detect the attack edges, the key is to design a betweenness metric that satisfies the detecting property – under this metric, the attack edges have high betweennesses, and non-attack edges have low betweennesses. By analyzing the properties of existing betweenness

metrics, Chapter 4 regarded that the RWEB should satisfy the detecting property. Chapter 4 then designed RSC to compute RWEBs of edges under malicious interference. Evaluations on real world and synthetic network topologies confirmed that the betweenness values computed by RSC indeed satisfied the detecting property. By regarding edges of high betweennesses as attack edges, the attack edges can be detected. Chapter 4 then provided an example showing that, by using RSC to detect attack edges, the accuracy of an existing SSD algorithm was significantly improved. This confirmed the effectiveness of RSC and its potential in creating accurate SSD algorithms for unauthorized DCSs.

The algorithms designed and findings made in this dissertation can effectively address the false result attack and the Sybil attack. Hence, this dissertation has made a stable contribution to ensure the application and development of DCSs.

In the future, this dissertation will be extended to two directions. The first is to design Sybil resisting mechanisms for a more general social network model. For a DCS, the existing SNM considers that honest nodes gather in a single cluster. However, in real DCSs, honest nodes may be divided into multiple clusters. Sybil resisting mechanisms for these DCSs should be designed. The second direction is to find flexible methods to combine SNM with DCSs. SNM has been regarded as a promising tool to consolidate securities of DCSs. However, most existing DCSs have their own connection protocols for realizing certain functionalities, and the portion of DCSs that satisfy SNM among all DCSs is still small. Hence, it is necessary to find flexible ways that enable DCSs to benefit from SNM without affecting their functionalities.

Bibliography

[1] Facebook. *http://www.facebook.com*.

[2] Skype. *http://www.skype.com*.

[3] Ian Foster. What is the grid? a three point checklist. Technical report, http://www-fp.mcs.anl.gov/ foster/Articles/WhatIsTheGrid.pdf, June 2002.

[4] David P. Anderson. Boinc: A system for public-resource computing and storage. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[5] SETI@Home. *http://setiathome.ssl.berkeley.edu/*.

[6] Mo Zhou, Yafei Dai, and Xiaoming Li. A measurement study of the structured overlay network in p2p file-sharing systems. *Adv. MultiMedia*, 2007:10–10, January 2007.

[7] Katayoun Sohrabi, Jay Gao, Vishal Ailawadhi, and Gregory J Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7:16–27, 2000.

[8] Elizabeth M. Royer and C.-K. Toh. A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications*, 6:46–55, 1999.

[9] Dongyu Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pages 367–378, New York, NY, USA, 2004. ACM.

[10] Shanyu Zhao, Virginia Lo, and Chris GauthierDickey. Result verification and trust-based scheduling in peer-to-peer grids. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 31–38, Washington, DC, USA, 2005. IEEE Computer Society.

[11] Philippe Golle and Ilya Mironov. Uncheatable distributed computations. In *CT-RSA 2001: Proceedings of the 2001 Conference on Topics in Cryptology*, pages 425–440, London, UK, 2001. Springer-Verlag.

[12] J. Douceur. The sybil attack. *Peer-to-Peer Systems*, pages 251–260, 2002.

[13] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S. Wallach. Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, 2002.

[14] G. DANEZIS, C. LESNIEWSKI-LAAS, M.F. KAASHOEK, and R. ANDERSON. Sybil-resistant DHT routing. *Lecture notes in computer science*, pages 305–318, 2005.

[15] Gheorghe Cosmin Silaghi, Filipe Araujo, Luís Moura Silva, Patrício Domingues, and Alvaro E. Arenas. Defeating colluding nodes in desktop grid computing platforms. *J. Grid Comput.*, 7(4):555–573, 2009.

[16] Newso James, Elaine Shi, Dawn Song, and Adrian Perrig. The sybil attack in sensor networks: analysis & defenses. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*, IPSN '04, pages 259–268, New York, NY, USA, 2004. ACM.

[17] Nguyen Tran, Jinyang Li, Lakshminarayanan Subramanian, and Sherman S.M. Chow. Optimal sybil-resilient node admission control. In *The 30th IEEE International Conference on Computer Communications (INFOCOM 2011)*, pages 3218–3226, Shanghai, P.R. China, 4 2011.

[18] H. Yu, P.B. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *IEEE Symposium on Security and Privacy*, pages 3–17. Citeseer, 2008.

[19] Chris Lesniewski-Laas. A sybil-proof one-hop dht. In *Proceedings of the 1st Workshop on Social Network Systems*, SocialNets '08, pages 19–24, New York, NY, USA, 2008. ACM.

[20] Chris Lesniewski-Laas and M. Frans Kaashoek. Whanau: A sybil-proof distributed hash table. In *NSDI*, pages 111–126, 2010.

[21] Luis F. G. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, CCGRID '01, pages 337–346, Washington, DC, USA, 2001. IEEE Computer Society.

[22] Wenliang Du, Jing Jia, Manish Mangal, and Mummoorthy Murugesan. Uncheatable grid computing. In *In 24th IEEE International Conference on Distributed Computing Systems*, pages 4–11, 2004.

[23] Cécile Germain, Gilles Fedak, Vincent Néri, and Franck Cappello. Global computing systems. In *LSSC '01: Proceedings of the Third International Conference on Large-Scale Scientific Computing-Revised Papers*, pages 218–227, London, UK, 2001. Springer-Verlag.

[24] C. S. Collberg and C. Thomborson. Watermarking, tamper-proofing, and obfuscation - tools for software protection. In *Software Engineering, IEEE Transactions on*, volume 28, pages 735–746, 2002.

[25] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 1–18, London, UK, 2001. Springer-Verlag.

[26] Ben Lynn, Manoj Prabhakaran, and Amit Sahai. Positive results and techniques for obfuscation. In *EUROCRYPT*, pages 20–39, 2004.

[27] Kanna Shimizu, Daniel Brokenshire, and Mohammad Peyravian. Cell broadband engine support for privacy, security, and digital rights management applications. *White paper*, 2005.

[28] Microsoft Corporation, Bennet Yee, Bennet Yee, J. D. Tygar, and J. D. Tygar. Secure coprocessors in electronic commerce applications. In *In Proceedings of The First USENIX Workshop on Electronic Commerce*, pages 155–170, 1995.

[29] D. Pham, S. Asano, M. Bolliger, M. N. Day, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Riley, D. Stasiak, M. Suzuoki, M. Wang, J. Warnock, S. Weitzel, D. Wendel, T. Yamazaki, and K. Yazawa. The design and implementation of a first-generation cell processor. *In Proc. ICICDT 2005*, pages 49 – 52, 2005.

[30] Amazon. *http://www.amazon.com*.

[31] N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-resilient online content voting. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 15–28. USENIX Association, 2009.

[32] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. A survey of dht security techniques. *ACM Comput. Surv.*, 43:8:1–8:49, February 2011.

[33] Brian Neil Levine, Clay Shields, and N. Boris Margolin. A survey of solutions to the sybil attack. *Tech report, University of Massachusetts Amherst, Amherst, MA, October 2006.*, 2006(2006-052):052, October 2006.

[34] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[35] A. Hernando, D. Villuendas, C. Vesperinas, M. Abad, and A. Plastino. Unravelling the size distribution of social groups with information theory in complex networks. *The European Physical Journal B - Condensed Matter and Complex Systems*, 76(1):87–97, 2010.

[36] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[37] Ling Xu, Chainan Satayapiwat, Hiroyuki Takizawa, and Hiroaki Kobayashi. Resisting sybil attack by social network and network clustering. In *SAINT*, pages 15–21, 2010.

[38] Daniele Quercia and Stephen Hailes. Sybil attacks against mobile users: friends and foes to the rescue. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 336–340, Piscataway, NJ, USA, 2010. IEEE Press.

[39] George Danezis and Prateek Mittal. Sybilinfer: Detecting sybil nodes using social networks. In *NDSS*, 2009.

[40] László Lovász and Peter Winkler. *Mixing of random walks and other diffusions on a graph*, pages 119–154. Cambridge University Press, New York, NY, USA, 1995.

[41] MEJ Newman. A measure of betweenness centrality based on random walks. *Social networks*, 27(1):39–54, 2005.

[42] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, March 1977.

[43] Michelle Girvan and M. E. J. Newman. Community structure in social and biological networks. *PROC.NATL.ACAD.SCI.USA*, 99:7821, 2002.

[44] Linton C Freeman, Stephen P Borgatti, and Douglas R White. Centrality in valued graphs: A measure of betweenness based on network flow. *Social Networks*, 13(2):141–154, 1991.

[45] Yih-Chun Hu, David B. Johnson, and Adrian Perrig. Sead: secure efficient distance vector routing for mobile wireless ad hoc networks. *Ad Hoc Networks*, 1(1):175–192, 2003.

[46] Tao Wan, Evangelos Kranakis, and P. C. Oorschot. S-rip: A secure distance vector routing protocol. In *In Proc. of Applied Cryptography and Network Security (ACNS'04*, pages 103–119. Springer, 2004.

[47] Hedrick. Rfc 1058. Technical report, Internet Engineering Task Force, 1988.

[48] Lada A. Adamic and Natalie Glance. The political blogosphere and the 2004 u.s. election: divided they blog. In *LinkKDD '05: Proceedings of the 3rd international workshop on Link discovery*, pages 36–43. ACM Press, New York, NY, USA, 2005.

[49] A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[50] NetworkX. *http://networkx.lanl.gov/contents.html*.

[51] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290, 1959.

[52] Bimal Viswanath, Ansley Post, Krishna P. Gummadi, and Alan Mislove. An analysis of social network-based sybil defenses. *SIGCOMM Comput. Commun. Rev.*, 40:363–374, August 2010.

[53] A. A. Pirzada and C. McDonald. Trust establishment in pure ad-hoc networks. *Wirel. Pers. Commun.*, 37:139–168, April 2006.

[54] Ling Xu, Ryusuke EGAWA, Hiroyuki TAKIZAWA, and Hiroaki KOBAYASHI. A network clustering algorithm for sybil-attack resisting. *IEICE Transactions, special section, Parallel and Distributed Computing and Networking*, E94-D,No.12:2345–2352, 2011.

[55] Christos Gkantsidis, Milena Mihail, and Amin Saberi. Random walks in peer-to-peer networks: algorithms and evaluation. *Perform. Eval.*, 63:241–263, March 2006.

[56] Laurent Massoulié, Erwan Le Merrer, Anne-Marie Kermarrec, and Ayalvadi Ganesh. Peer counting and sampling in overlay networks: random walk methods. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, PODC '06, pages 123–132, New York, NY, USA, 2006. ACM.

[57] Nabhendra Bisnik and Alhussein A. Abouzeid. Optimizing random walk search algorithms in p2p networks. *Comput. Netw.*, 51:1499–1514, April 2007.

[58] Marc Bui, Thibault Bernard, Devan Sohier, and Alain Bui. Random walks in distributed computing: a survey. In *Innovative Internet Community Systems 2004. Revised Papers, Lecture Notes in Computer Science*, pages 1–14, Guadalajara, Mexico, 2004. Springer.

[59] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of*

the 16th international conference on Supercomputing, pages 84–95, New York, NY, USA, 2002. ACM.

[60] William Stallings. *Cryptography and Network Security: Principles and Practice*. Pearson Education, 3rd edition, 2002.