Allocation schemes, Architectures, and Policies for Collaborative Port Scanning Attacks

Yu Zhang, Bharat Bhargava
Department of Computer Science, Purdue University, West Lafayette, USA
Email: {zhangyu, bb}@cs.purdue.edu

***Abstract*—Most network attackers perform port scanning individually, without synchronization, to find victim hosts. Such port scanning schemes suffer from two problems: first, there are too many duplicate scannings and too much contention among different port scanners; second, a complete port scanning takes a long time to finish.**

In this paper, we present a fast DHT-based collaborative port scanning scheme that aims to eliminate duplicate scanning, minimize contention, and significantly increase the scanning speed. In collaborative attacks, attackers communicate and collaborate with each other to launch much more powerful attacks. In the DHT-based collaborative port scanning scheme, attackers collaborate to search the network for ports that could be exposed to attacks.

We propose different collaborative scanning strategies and analyze their advantages and disadvantages. We discuss the static, dynamic, and hybrid target selection and allocation schemes. We present the algorithm details and discuss the stop and revisit policy for the collaborative port scanners. We conduct experiments to evaluate the performance and overhead of the collaborative port scanning strategies. Experimental results suggest that the proposed collaborative port scanning system significantly increases the efficiency of port scanning and provide insights into many design and implementation issues.

***Index Terms*—Port Scan, Collaboration, Network Security.**

I. Introduction

Attackers employ various technologies to launch attacks, such as Denial-of-Service (DoS), BotNet, Worm, and Virus, *etc*. The first step of these attacks is to discover vulnerable victim hosts.

Nearly all attackers perform port scanning to find vulnerabilities on victim hosts. Most existing fast-replicated viruses and worms [6],[39],[40] perform port scanning to discover and infect targets. Hence, it is crucial to study port scanning and explore whether the latest advances in technologies have changed the horizon of port scanning, including how to perform port scanning, expedite scanning speed, conduct port scanning from multiple machines, and defend against modern port scanners.

Different network protocols employ different ports. Vulnerabilities exist in all protocols. Hence, to gather information completely, port scanners have to perform scanning for a large number of ports. The size of the port space is 65535 [13]. Ports 0 to 1023 are well-known ports, ports 1024 to 49151 are registered ports, and ports 49152 to 65535 are dynamic or private ports.

Port scanners must run extremely fast. Port scanners have employed sophisticated techniques to expedite port scanning. For example, worms can search vulnerabilities on a commonly used port (e.g., port 21 used by FTP, and port 443 used by HTTPS). However, a typical complete port scan is time-consuming. For example, a 65,536-port UDP scan for one target host could take more than 18 hours [1].

Attackers typically perform port scanning independently, without coordination, to find victim hosts. If port scanning software packages are run on multiple machines without coordination, their search spaces will overlap significantly. The overlap causes reduction in the performance of the scanning. The network connections used by the port scanners could get congested. The buffer size of the network software may not be large enough to hold all the incoming data. The processing speed of the computer may not be enough to analyze responses from all the networks. After all scanning activities end, all the computers involved in the scanning must communicate to each other and finalize the search results. Problems arise when their results differ. Such differences are hard to analyze, due to the fast-changing nature of computer networks.

As defense technologies evolve, port scanners that exhibit unusual network behaviors, such as sending requests to all IP addresses in a Class B network, are more likely to be detected. Such detection will likely disable the machine performing the scanning immediately and trigger chained detections of all other machines involved. Given the fact that virtually all networks are protected by firewalls, filters, and monitors, a simple deployment of identical port scanning software packages to all computers involved in the scanning is not acceptable.

A key observation to the above deployment plan is that there is a lack of collaboration among the port scanners. In the rest of the paper, we use collaboration and coordination interchangeably.

A simple increase in the number of port scanners creates too much duplicate work, increases the power of the whole attack incrementally, and introduces overhead on analyzing, comparing, and resolving the conflicts in the results. Therefore, we need a smarter deployment plan which makes full use of all scanners involved, avoids performing duplicate actions to the maximum extent, and synchronize actions of participating nodes properly and efficiently.

We propose a smart and efficient deployment plan of port scanner software packages to multiple computers, the DHT-based Collaborative Port Scanning. To address the issues above, our deployment plan employs the Distributed Hash Table (DHT) to speed up the scanning, avoid duplicate scanning and contention, and efficiently process and summarize

the results from multiple computers. The key idea is to perform DHT lookups on the target host before initiating any scanning activity. In general, the idea can be extended to other collaborative attacks as well.

DHTs [41] are distributed systems that provide essential functionalities of hash tables (HT). In HTs, one can insert items and query whether a particular item exists. For each stored item, an associated value can be retrieved in the HT. Similarly, in DHTs, one can insert items and query for the existences of items. (Key, Value) pairs can be stored in DHTs as well.

The difference between a DHT and a conventional HT is that the stored items are distributed over multiple computers. The key advantages of the DHT include:

1) *Efficiency:* DHT is designed to store information and perform lookups efficiently.
2) *Scalability:* DHT is designed to scale to thousands of computers.
3) *Robustness:* Most DHTs can let participating computers to join or leave at any time, and gracefully handle computer failures.
4) *Distribution:* Items stored in DHTs are distributed over a large number of computers. No central server is needed to answer queries.

Numerous application have employed DHT in the past, such as BitTorrent and Emule [43],[44].

DHT fits the collaborative port scanners because it allows efficient lookups of IP addresses (or IP address and port number pairs). In our deployment plan, each local port scanner double-checks an IP address in the DHT before the actual scanning, therefore avoids problems on duplicate scanning and contention. The DHT database serves as a result repository. It can store scanned results as the (IP address, scan result) pairs. Although DHT has associated overhead in insertions and lookups, it provides higher efficiency by avoiding duplicate scanning, contention, and unnecessary data analysis. Hence, the DHT-based collaborative port scanning scheme significantly improves over uncoordinated and unsynchronized scanning.

The rest of the paper is organized as follows. Section II discusses related work. In Section III, we discuss existing port scanners. In Section IV, we present the DHT-based collaborative port scanning scheme. In Section V, we conduct experiments to measure the performance of the proposed scheme. Section VI concludes our paper.

II. Related Work

Port Scan. Port scanners employ various techniques. In a SYN Scan, the scanner produces its own IP packet and sends TCP SYN packets to victim hosts and analyzes the responses. In a UDP scan, the scanner sends UDP packets to victim hosts and checks whether ICMP port unreachable messages are received afterwards.

Port scanning can be performed on multiple ports. Some scanners perform the scanning in two-iterations. These scanners scan with one technique, e.g., SYN scan, first before scanning the un-denied ports with other techniques. For instance, the famous NMAP scanner [1] uses the two-iteration approach when executed with the -SUV option.

Ref. [37] discussed how to detect coordinated port scans. However, the author mainly focuses on the detection and did not provide details of how to coordinate the individual port scans.

Port scanner needs to scan a large number of ports, as discussed in Ref. [13]. Example ports include port 7 for echo, port 21 for FTP, port 22 for SSH, port 23 for Telnet, port 25 for SMTP, port 80 for HTTP, port 79 for finger, port 110 for POP3, port 139 for NetBIOS, port 143 for IMAP, port 443 for HTTPS, and port 53 for DNS.

Scan Strategy. Researchers have proposed various scanning algorithms for port scanner, including: *naive random scanning*, in which the port scanner chooses a random address uniformly from the IPv4 address space [10]; *localized scanning*, in which the port scanner scans a local IP address (e.g., addresses that are in the same subnet as the port scanner) with a high probability p and scans a random address with a low probability (1-p) each time [38]; *importance scanning*, in which the port scanner assumes that the vulnerable hosts are unevenly distributed and such distributions are obtainable [23]; *self-learning scanning*, in which the port scanner estimates the distribution of the vulnerable hosts [21]. In contrast, our approach can keep the exact information. The self-learning worm estimates the distribution of vulnerable hosts based on information about the infected hosts. If we employ a combination of scan and attack strategy, we may try a new strategy (e.g., exploiting a new vulnerability) to avoid making a premature decision. Another observation is that infection takes more time than scanning. It does not happen instantaneously, and prevents faster spreading; *hit-list scanning*, in which the port scanner uses an existing list. e.g., BGP routing table list, social network list, etc., to look for vulnerable hosts [14]; *permutation scanning*, in which the port scanner could determine whether a host was infected and change scan targets [14]; *sampling scanning*, in which the port scanner samples the target network before spreading to the corresponding subnet [34]; and *passive scanning*, in which the port scanner analyzes the network traffic passively, etc.

Coordination. Port scanners can collaborate with each other and perform much more efficient reconnaissance. Staniford et al. [14] discussed the Warhol worm, which propagates extremely fast by self-coordination with both hit-list and permutation scanning. Wiley [20] described an abstract distributed and collaborative worm Curris Yellow. Gates [37] discussed possible collaborations in port scans. Wang et al. [32] described an advanced peer-to-peer Botnet. The distributed.net [47] used distributed computing to break ciphers. Our work discusses specific issues of the DHT-based scheme, proposes different allocation strategies, and illustrates the scanning architectures and policies.

IPv6 Scan. Yang [12] discussed how to defend worms in IPv6 networks. Bellovin et al. [31] presented worm propagation strategies for IPv6 networks. Kamra et al. [35] proposed a DNS-scan method that can achieve high spread rates in IPv6 networks.

Worm Scanning. Ref. [33] discusses characteristics of

worms, including protocol, amount of payload and scanning strategy, etc. Ref. [22] talks about the performance and models of worm propagations. The authors talk about the local preference scans. If there are multiple attackers starting from multiple sources, local preference scans will be much more powerful. Ref. [27] describes a class of worms that target network systems such as routers. Ref. [34] discusses how to minimize the number of scans required to infect hosts. Wagner et al. [33] presented characteristics of worms, including protocol, size of the payload, and scanning strategy, etc. Zou [22] et al. analyzed the performances of different propagation strategies. Voyiatzis et al. [27] described a class of worms that target network components such as routers. Vojnovic et al. [34] discussed how to minimize the required number of scans to infect hosts. Storm Worm [6], [7] used the Distributed Hash Table (DHT) protocol based on Kademlia [8] to control infected nodes. Chen et al. [10] proposed the Analytical Active Worm Propagation (AAWP) model. Zou et al. [39] proposed the epidemiological two-factor model. Dagon et al. [18] discussed taxonomy of Botnets. Zhang et al. [5] discussed a Fibonacci model of worm propagation.

Defense. Wu et al. [24] proposed a worm detection architecture for various worm scanning techniques. Twycross et al. [25] built a virus throttle program that can detect the port scanner based on their abnormal network behaviors. Jung et al. [29] developed the Threshold Random Walk (TRW) algorithm to identify malicious remote hosts. Kumar et al. [36] presented the analysis of the Witty Worm and inferred about the IP address where the Witty Worm was released. Staniford et al. [30] described Spice, a port scanner that can detect stealthy scans.

III. Issues on Port Scanning

Most attacks include the *reconnaissance* step, in which attackers explore and discover victim hosts for vulnerabilities and important information for launching attacks, including operating system and firewall status. Port scanners are regularly used to perform such activities.

A. Conventional Port Scanners

In a *port scan*, attackers scan a number of listening ports on the victim host. This method guarantees that all known vulnerabilities to attackers on the victim host can be discovered, i.e., there is no false negative. However, an exhaustive search is time-consuming. On the other hand, in a *port sweep*, attackers scan a particular port on the a large number of victim hosts. Port sweep can reduce the size of search space significantly, but could ignore vulnerabilities on un-searched ports. It is clear that the optimal strategy is to scan only the common or vulnerable ports. Such strategy is difficult to achieve in practice. It is not uncommon to find attackers that employ a combination of both methods. Note that in a *partial scan*, attackers can only scan the ports that match the vulnerabilities that the attackers want to take advantage of. E.g., if the attacker could launch FTP and HTTP attacks, ports 21 and 80 could be the only ports that the attacker scans. If there are multiple collaborative attackers, one attacker can provide vulnerability database such as the FTP and HTTP ports mentioned above, another attacker can perform optimal scanning according to the available vulnerability database.

For each port scan attempt, the result can be:

1) *listening*: the scanned port is actively listening. E.g., provided that the victim host tries to accept the TCP connection request, a successful TCP SYN scan receives a SYN-ACK packet from it.
2) *not listening*: the scanned port is not listening. E.g., if the victim host does not listen on a particular UDP port, a UDP scan directed to that port receives an ICMP port unreachable packet.
3) *unknown*: there is no response from the victim host. The IP packets between the scanner and the victim host may be lost on the way, filtered by firewall, or blocked by the anti-virus software.

Successful scans can yield promising results, including the operating system of the target host, the protocol suite in use, and the open ports, etc. Note in case 3), victim hosts may not trust the machine running the port scanner for a number of reasons, including IP address not recognized, host not residing on the same LAN, etc. In such cases, a collaborative scan launched from hosts that are trusted by victim hosts may be successful. The information gathered by the trusted hosts can be passed to other malicious computers.

B. Detection of Port Scanners

Security monitors that could detect port scanning activities normally employ simple rules to label potential port scanning activities. E.g., the monitors can check whether there are a large number of probes (denote it as m) within a limited time period (denote it as n seconds) from a particular machine. Note n is normally set to a small number to reduce the burden of the Intrusion Detection Systems (IDS) due to their limited ability to log and analyze network traffic. Researchers have proposed new techniques for detecting port scanning activities, such as advanced techniques that employ machine learning and probabilistic packet inspection [29]. Port scanners that choose scan targets randomly are more likely to be detected because of the large-scale network-indicators generated [17].

C. Collaborative Port Scanners

Individual port scanners have limited power because they usually employ a specific technology. Ironically, they are more likely to get detected because they scan all targets individually and generate excessive network traffic. Collaborative port scanners can perform the work together. E.g., they can vary the port scanning technologies, the port scanning locations, the methods to choose scan targets, and the ways to divide work among themselves. We discuss possible scanning technologies and methods below.

C1) Port Scanning Technologies

Collaborative port scanners can choose from a variety of port scanning technologies, including but not limited to [26], [29], [30]:

1) *Connect scan*: The port scanner employs system call connect() to scan target hosts. This scan does not require

special privileges. After a TCP connection is established to the victim host, the port scanner sends a RST packet to close the connection. One drawback of this scan is that the established connections are logged and easily noticed by security monitors and software packages.

2) *Application scan*: The port scanner employs particular application-layer protocols and sends requests according to the protocol-specifics. Example protocols include HTTP, FTP, and DNS. If the victim host responds to such application-layer requests, the port scanner classifies the corresponding ports as active.
3) *Ident scan*: The port scanner connects to the victim host, and uses a vulnerability in the ident protocol to retrieve usernames on the victim host. E.g., if the port scanner connects to a HTTP server, the ident protocol can be used to look up the username running it.
4) *SYN scan*: The port scanner crafts its own TCP packet. The scanner first sends a SYN packet to the victim host. The victim host responds with a SYN/ACK packet. The scanner records this response and classifies the scanned port as listening and accepting incoming connections. The scanner could then send a reset (RST) packet to end the scan. Since the SYN scan does not establish a full TCP connection, the victim hosts will not run out of buffer space for accepting incoming connections.
5) *UDP scan*: The port scanner crafts its own UDP packet. The scanner sends an arbitrary UDP packet to the victim host. If an ICMP port unreachable message is received afterwards, the scanner knows that the port is not active. However, reply packets might be dropped on the network route. Certain network security monitors and anti-virus software packages may filter out the UDP packets. Therefore, this scanning technology can be unprecise.
6) *ACK scan*: The port scanner crafts its own probe packet and set the ACK flag. The scanner sends the probe packet to the victim host. If an ICMP unreachable message is received or there is no reply, the port scanner confirms that the probe packet is filtered by firewall or security software packages. Therefore, ACK scan is used to detect whether a particular network link is guarded by firewall or security softwares. If the network link is unfiltered, the victim host will return an RST packet.
7) *FIN and Null scan*: The port scanner produces surreal scenarios and analyzes the responses from the victim hosts. In a FIN scan, the scanner sends a FIN packet to the victim host. If there is no reply from the victim host, the scanned port on the victim host is classified as open or filtered, because a closed port would send an RST packet. Similarly, in a Null scan, the port scanner produces a TCP packet that does not have any flag, and sends it to victim hosts. A lack of response suggests that the target port is either open or filtered.
8) *Cloaked scan*: In cloaked scans, it is very difficult for defenders to figure out the identity of the scanners. Network devices, firewalls, security software packages, and servers can log potentially malicious activities (e.g., a connection without data transfer) and analyze them to find the scanners. Port scanners can use cloaked scans in such cases. Example cloaked scans include: a) *proxy scan*: in which the victim host sees a proxy machine rather than the attacker; b) *fragmented packet scan*: in which the port scanners send fragmented packets that can be combined together at the destination; and c) *implementation-flaw scan*: in which the port scanners exploit implementation flaws in the victim host to perform scanning. E.g., the old predictable IP ID sequence number bug [46] (now fixed) can be employed to do port scanning.
9) *Multi-cast scan*: The port scanners can send packets to a multicast address in this case. The packet is then directed to a large number of victim hosts. The port scanner can fake the source IP addresses so that responses can be directed to other collaborative attackers.

C2) Target Selection Methods Collaborative port scanners can choose from a variety of target selection methods:

1) *Naive scanning*: The port scanner chooses the next IP address to scan according to an uniform distribution. Code red and Slammer worms employed this method.
2) *Local scanning*: The port scanner gives priority to local IP addresses. More specifically, with probability p the port scanner chooses to scan an IP address that shares the same first x bits (x can be any number from 1 - 31) with it, and with probability (1-p) it chooses to scan a random IP address.
3) *Importance scanning*: The port scanner assumes that the vulnerable hosts are unevenly distributed, hence important IP addresses should be scanned first. Ref. [23] proposes the importance-scanning method, assuming that the vulnerable host distributions exist and are obtainable. Ref. [21] proposes the static importance-scanning strategies and assumes that keeping information about uninfected hosts is realistic.
4) *Sequential scanning (nearest neighbor scanning)*: The port scanner chooses to scan the next IP address in the lexicographical order.
5) *Hit-list scanning [14]*: The port scanner employs an existing list of IP addresses and scans those addresses first. Such tables may be easily obtained from multiple sources, such as routing tables and social network profiles. In particular, the hitlist that has all the addresses in the BGP routing table is very easily obtained and effective.
6) *Sampling scanning*: The port scanner can choose to scan the representatives of subnets. After successful scans, the port scanner tries to infect the victim hosts and then start new scanning from the newly infected hosts.
7) *Passive scanning*: The port scanner does not send scanning packets. Instead, it collects and analyzes the network traffic that pass through it.

IV. Background on DHT

DHTs [41] are distributed systems that provide essential functionalities of hash tables. In hash tables (HT), one can insert items and query whether a particular item exists. For each item stored, a value associated with the item can be

retrieved in the hash table. Similarly, in DHTs, one can insert items and query for the existence of items. (Key, Value) pairs can be stored in DHTs too.

The difference between DHT and conventional HT is that the stored items are distributed over multiple distributed computers. The key advantages of the DHT include:

1) *Efficiency:* DHTs are designed to store information and perform lookups efficiently.
2) *Scalability:* DHTs are designed to be able to involve thousands of computers.
3) *Robustness:* Most DHTs allow participating computers to come and go, therefore can gracefully handle computer failures.
4) *Distribution:* Information stored in DHTs are distributed over a large number of computers. No central server is needed to answer queries.

Numerous application have employed DHT in the past, such as [43],[44].

V. DHT-BASED COLLABORATIVE PORT SCANNERS

The fundamental problem for port scanners is to find vulnerabilities on all ports of target hosts, build exhaustive vulnerability database, and prepare for the launch of effective attacks against target hosts. As discussed in Section I, port scanning can start from multiple sources instead of only one. The latter is assumed in current research. Multiple port scanners might perform duplicate scanning or cause contention.

Furthermore, as discussed in Section III-C, port scanners on different machines can employ different technologies to scan the target hosts. It is not only inefficient to let all port scanners try all possible scan methods, but such exhaustive searches are also likely to trigger the alarms of defense systems.

To avoid contention among port scanners and increase scanning speed and power, the collaborative port scanning scheme must define clear work allocation methods for all participating scanners, avoid generating excessive network traffic or leaving traces for tracebacks, and specify when to stop scanning for the scanners. We discuss these issues below.

A. Static and Dynamic Allocation of Targets

A number of attackers can perform the port scanning simultaneously to make much faster progresses. In this scenario, a number of attackers can divide the work to scan a large number of victim hosts. There are two ways to divide the work:

A1) Static Allocation

The Static Allocation (SA) scheme avoids the duplicate work discussed above. In a SA scheme, the target address space is divided to all collaborative port scanners before the launch of the actual attack. Port number and IP address combinations, scanning technology, and vulnerability checking methods are divisible as well in this scheme.

Each port scanner gets its own allocation of the target space, technology (to use), and vulnerability (to check) list. Without loss of generality, we only discuss the allocation methods for the target space.

Collaborative port scanners have a number of ways to define the SA policy to divide the work, i.e., ways to divide the large target address space. Examples include:

1) *Divide the address space by hosts.*
   In this policy, each collaborating port scanner will be responsible for all ports on particular hosts. There are two methods to divide the addresses:
   a) *random*: This policy divides the target address space randomly to individual port scanners.
   b) *sequential*: This policy divides the target address space sequentially to individual port scanners. Each scanner will be responsible for a chunk of the huge IP address space.
2) *Divide the address space by port numbers.*
   In this policy, each collaborating port scanner will be responsible for the same port on all hosts. There are two methods to divide the addresses as well : random and sequential.

Host-based approaches can suffer from scanning detection employed at host-level, because the scanning will cover all the ports on the target host and this could easily trigger the alarm of the detection system. Port-based approaches, on the other hand, can avoid such detection. However, the network-level detection tools may still find the port-based attacks.

The SA scheme does not address a number of issues, including node failure and dynamic node joining and leaving. Note that in real attack scenarios, such as the worm scanning and propagation, newly infected hosts might join the existing attackers. Also, not all port scanning activities are equal. For instance, routers and firewalls only filter packets from particular sources. Hence, port scanning packets from one scanner may be filtered, while the packets from another may go through.

Moreover, port scanning can be done in different ways. One can initiate the port scanning with different power, such as packet sending rates. One can have unsuccessful scans due to network data loss and jittering. One can get active defense from defenders of the target systems and could even get detected and physically disabled by them. SA scheme fails to address these issues either.

A2) Dynamic Allocation

The Dynamic Allocation (DA) scheme does not pre-allocate target spaces and allows the attackers to divide the work on the fly. In this scheme, attackers can communicate with each other to dynamically determine the next hosts to scan. A key advantage of communication between attackers is that the scanning space can be constantly updated.

In the state-of-the-art port scanners, e.g., the hit-list based worm scanners, the hit-list is divided by half each time using a top-down allocation approach when a new propagation is successful. In contrast, the DA scheme allows the scanning space to be constantly updated.

In the DA scheme, one can develop distributed lookup tables to query whether a particular IP address has been scanned/infected or not.

A3) Hybrid Allocation

The Hybrid Allocation (HA) scheme combines the SA and DA schemes together. If the number of target hosts is large

and the number of available attackers is not small, the system can divide the target hosts statically in the first step. The statically divided hosts are then assigned to different groups of attackers. The attackers for particular groups use the DA scheme to generate scan targets.

For instance, to scan all target hosts a country, one can divide them by states, and allocate hosts in different states to different groups of attackers. The HA scheme is most useful when there are lots of target hosts and attackers.

B. Synchronization of collaborative port scanners

Based on the above discussion, in order to synchronize the actions of collaborative port scanners, we need to develop a dynamic or hybrid allocation scheme that allocates the scanning targets to individual port scanners. The DA or HA scheme must be extremely efficient so that it can respond to multiple requests from a large number of collaborative port scanners. During the on-the-fly target allocation, the DA scheme needs to make sure that two conditions are met:

1) No two port scanners will be scanning the same IP address.
2) A port scanner will not be scanning any IP address that has already been scanned by another port scanner.

To facilitate our discussion, we define the status of an IP address based on whether it has been scanned.

Definition 1. Port Scanning Status (PSS)

The Port Scanning Status (PSS) of an IP address is a two-bit number that indicates its the scanning information. More specifically, the PSS of an IP address *i* can be one of the following:

1) *00* — denotes that the IP address has never been scanned by any collaborating port scanner;
2) *01* — denotes that the IP address is currently being scanned by a collaborative port scanner;
3) *10* — denotes that the IP address has already been scanned by some port scanner.
4) *11* — denotes that the IP address has already been attacked by some attacker based on the scanning results.

Definition 2. Degree of Collaboration (DC)

The Degree of Collaboration (DC) for a collaborative port scan is an integer that records the number of active collaborative port scanners. Since individual port scanners may join and leave at any time during a collaborative port scan, the number of active collaborative port scanners vary from time to time. Hence, the DC for a collaborative port scan at time tick t is :

DC_t = the number of active port scanners at time t.

Definition 3. Collaboration Architecture (CA)

Collaborative attackers need to communicate to each other over the network to synchronize their actions. In particular, collaborative port scanners need to synchronize their scanning activities on IP addresses.

To effectively communicate the PSS of an IP address and keep themselves updated about the DC, they need an efficient and robust distributed query system. The basic functionality of the query system is to store scanned results and provide real-time scanning status. Information like the scanned IP addresses, ports, and vulnerabilities are stored in the system.

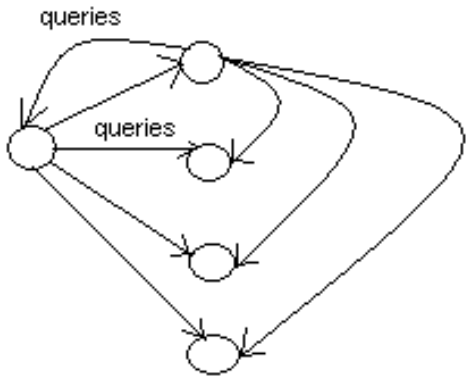

Fig. 1. The flooding architecture.

To facilitate discussion, we consider the case that only IP addresses are stored.

There are a number of possible architectures for this query system :

1) *Flooding architecture*: As shown in Fig. 1, in this architecture, each collaborating port scanner "floods", i.e., broadcasts messages to, all known collaborators to query the scanning status of a particular IP address. While robust against node failures, this architecture requires that each node stores its own scanning status information and incurs significant network overhead due to the huge amount of query traffic.
2) *Collaboration-server based architecture*: As shown in Fig. 2, in this architecture, each collaborating port scanner registers itself at a collaboration server dedicated to monitoring scanning status, and joins the collaborating port scanner group. If a collaborating port scanner stops the scanning activities, it will notify the collaboration server. The collaboration server is responsible for storing the scan status and results for all port scanners. each collaborating port scanner queries the collaboration server for real-time scanning status and makes decision on the next scan target. While efficient, the reliability of this architecture depends on that of the collaboration server. If the collaboration server has limited bandwidth, it will not able to handle the large amount of network traffic generated by individual port scanners.
   Moreover, the collaboration server is a single failure point. The collaborative port scan could not proceed if the server is down. Even if a new collaboration server can be established, a lot of efforts and time need to be spent in the recovery. The collaboration-server architecture is vulnerable to defense as well, e.g., the defenders can analyze the traffic patterns of the collaboration server, determine that it is acting as a communication and command center, and take it down to shut down the whole collaborative port scan. In the real world, researchers have proposed traffic analysis methods to defend against Botnets based on IRC channels [18],[19].
3) *Distributed architecture*: As shown in Fig. 3, in this architecture, the scanning status of all collaborating port

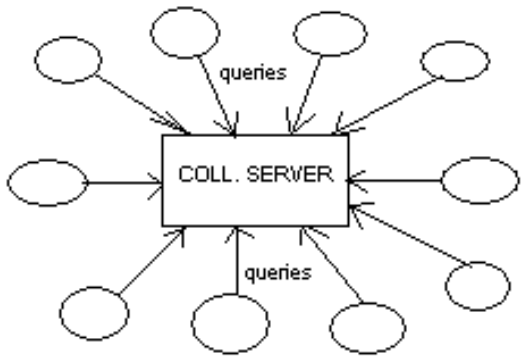

Fig. 2. The collaboration-server based architecture.

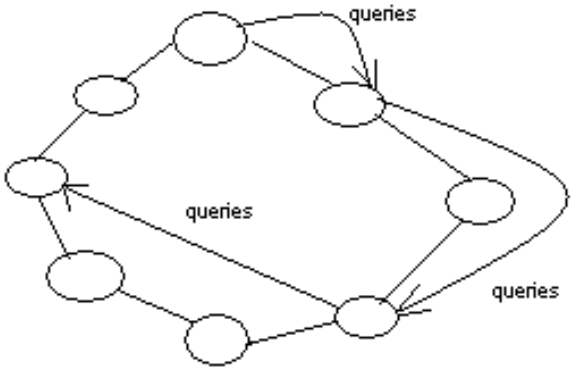

Fig. 3. The distributed architecture.

scanners are distributed over all the scanners. Therefore, each collaborating port scanner issues queries to the distributed information system based on themselves. The distributed information system acts as the efficient storage and query server, scales to a large number of nodes, and provides high reliability.

This architecture eliminates the concentration of information and network traffic on the collaboration server. Its efficiency is much higher than the flooding architecture. However, each port scanner has to both perform scanning and serve as a active node in the distributed information system.

4) *Hybrid collaboration architecture*: As shown in Fig. 4, in this architecture, there are two groups of collaborating attackers: the first group of them is the traditional port scanner group, and the other is the information group, i.e., the one responsible for the distributed information system discussed in the distributed architecture. The attackers from the first port scanner group query the attackers from the information group for IP address scan status. The attackers from the information group builds, indexes, and stores all scan status information efficiently. Attackers from the port scanner group view the distributed information system as a collaboration server discussed in 2).

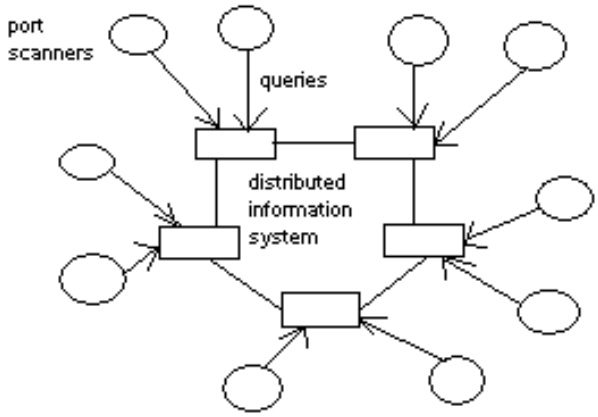

Fig. 4. The hybrid architecture.

The hybrid collaboration architecture combines the collaboration-server based architecture and the distributed architecture. Compared to the distributed architecture, the hybrid architecture relieves the port scanners from infrastructure issues, i.e., storing scanning status information and answering IP scan status queries. It specifies a dedicated group of attackers responsible solely for the distributed information system on scanning status. Hence, attackers do not have to balance their resources between the actual attacks and the infrastructure. By such task division and collaboration, attackers take advantage of the benefits of both collaboration-server based and the distributed architectures. Therefore, they are more likely to increase the efficiency and the scalability of their systems and launch much more powerful attacks.

C. *The DHT-based Contention-Avoidance Allocation Scheme*

*1) **Overview**:* We need a distributed port scanning system that can avoid duplicate scanning and contention among collaborative port scanners. Duplicate scanning and contention include simultaneous scanning of an identical victim host, generation of excessive network traffic on the same network link, and duplicate work of distributed port scanners, i.e., scanning the same port on the same victim host for an identical vulnerability.

The proposed DHT-based collaborative scanning scheme can elegantly avoid duplicate scanning and contention. The scheme incorporates the well-designed distributed lookup system, the DHT, that stores scanning status information and answers queries from collaborative port scanners. The DHT provides distributed look-up services.

Based on the discussion in Section V-B, the collaborative port scanning scheme employs the distributed/hybrid architectures because they provide higher efficiency and scalability.

Traditional port scanners send probing packets and analyze responses from victim hosts. In the collaborative port scanning scheme, each collaborative port scanner queries the DHT before each scanning. Note besides DHT, there are other candidates for the distributed information storage and query

system, such as the Big Table [2]. If false positive can be tolerated, Bloom Filter [4] can help with the query system as well.

*2) **The DHT-based scanning algorithm**:* Each collaborative port scanner runs the new DHT-based scanning algorithm in the proposed scheme.

The algorithm for the collaborative port scanner is presented in Algorithm 1. The collaborative port scanner randomly picks up an IP address and a port number, and checks if the IP address and port number combination has been scanned already.

If not, the port scanner performs port scanning activities using a randomly chosen scanning technology discussed in Section III-C, and records the scanned results. There are two important methods for the collaborative port scanner: the GET() method, responsible for checking the scan status for IP address and port number combinations, and the SET() method, responsible for recording the scanned results.

The algorithm for the GET() method in the scanner is presented in Algorithm 2. The GET() method processes the (IP address, port number) pair, and looks it up in the DHT to check its scan status. If there is a match, the GET() method returns a variable indicating that the (IP address, port number) pair has already been scanned. Otherwise the GET() method returns a variable indicating that the pair has not been scanned.

To perform look-ups in the DHT, the GET() method can employ RPC calling mechanisms. Note that DHT performance optimizations allow fast lookups. E.g., caching and the hybrid architecture discussed above can significantly reduce the lookup latency.

The algorithm for the SET() method in the scanner is presented in Algorithm 3. The SET() method processes the (IP address, port number) pair, and records its status in the DHT. Note the SET() method must take concurrency control issues into consideration because no concurrent GET() and SET() should be allowed on the same (IP address, port number) pair to ensure correctness of the whole system.

Although concurrent GET() accesses are allowed, no two SET() methods should be modifying the scanning status of the same (IP address, port number) at the same time. This problem is similar to the reader/writer lock problem: concurrent GET()s is allowed while concurrent SET()s and concurrent GET()s/SET()s are prohibited. The GET() and SET() methods can implement a reader/writer lock in this case to improve the lookup performance.

Another way to improve the performance is to lock only part of the DHT. By default, given an (IP address, port number) pair, the SET() method can request to lock certain IP address ranges instead of the whole IP range. E.g., if the SET() method requests to lock only the Class B network that the given IP address belongs to, other SET() methods can write to other class B network addresses, which improves the performance of the whole system.

Algorithm 1 The Collaborative Port Scanner

```
// Get an unscanned IP address and port number combination.
repeat
  // Do preprocessing works.
  //
  ip = ChooseIPAddressToScan();
  port = RandomlyPickUpAnPortNumber();

  // Check with the DHT to see if the IP address
  // and port number combination has been scanned already.
  // The GET() method returns NOT_SCANNED if the
  // IP address and port number combination has not been
  // scanned yet.
  scan_status = GET(ip, port);
until scan_status = NOT_SCANNED

// Perform the scanning activities. The scanning method
// is generated randomly from the scanning technology
// database, including the ones discussed in Section III-C.
scan_method = RandomlyPickUpScanningMethod();
scan_method.Send(probing packets, victim);
scan_method.Receive(responses, victim);
scan_method.scan_result = Analyze(responses);

// Record the result of the scanning to the DHT.
SET(ip, port, scan_method.scan_result);

// Return.
return OK;
```

D. Detection Avoidance

An effective way to detect traditional port scanners is to watch for abnormal network traffic patterns. As discussed in Section I, thresholds such as excessive number of pings within a certain time period can be set up to trigger alarms for port scanning activities. An obvious "solution" is to perform "stealth scans", e.g., perform scanning activities slowly for several months and gather the results. Such solution cannot get enough information quickly and is not desirable for the port scanners.

Collaborative port scanners can distribute the work among a large number of machines that are in different geographical areas, thus reduce the network traffic generated by individual port scanners and avoid detection. An optimal scanning strategy for detection avoidance is to "blend into the crowds", i.e., to mix scan traffic into normal network traffic and make it difficult for defenders to notice. E.g., smart scanning schemes that resemble the collaborative port scanners as web crawalers, bots, or spiders could successfully foil a large number of defense systems.

Moreover, by employing the distributed and the hybrid architectures discussed in Sec.V-B to distribute network traffic, collaborative port scanners can escape detection of intelligent defenders. Methods that analyze the network traffic using data mining techniques [19] to identify command centers of collaborating malicious computers will fail to locate collaborative port scanners.

Algorithm 2 The GET Method

```
// The GET() method :
// takes:
// (ip address, port number) pair as inputs; and
// returns:
// NOT_SCANNED : if the pair has not been scanned yet;
// SCANNED : if the pair has been scanned already.

Require: IP address and port number are correctly passed in
  as arguments.

  // Do preprocessing works.
  ip_port_pair = GeneratePair(ip, port);

  // Contact the DHT to read information.
  RPCCallBack = SetupRPCCall();
  RPCCallBack.Run();
  WaitForRPC();

  // Get the scanning status for the (IP address, port number)
  // pair.
  scanning_status = ProcessRPCResults();

  // Return.
  if scanning_status = 0 then
    return NOT_SCANNED;
  else
    return SCANNED;
  end if
```

Algorithm 3 The SET Method

```
// The SET() method :
// takes:
// (ip address, port number) pair and
// the scanned result as inputs;
// performs:
// DHT information updates.

Require: IP address, port number, and the scanned result are
  correctly passed in as arguments.

  // Do preprocessing works.
  ip_port_pair = GeneratePair(ip, port);

  // Request exclusive access for the DHT.
  // The granularity can be :
  // for the whole IP range (slowest);
  // for the whole Class A address (slow);
  // for the whole Class B address (medium);
  // for the whole Class C address (fast); or
  // no exclusive access (fastest).
  Lock(starting IP address, ending IP address);

  // Contact the DHT to write information.
  RPCCallBack = SetupRPCCall();
  RPCCallBack.Run();
  WaitForRPC();

  // Check if the update was successful.
  CheckSuccess();
Ensure: Update is successful.

  // Release the lock.
  Unlock(starting IP address, ending IP address);
Ensure: lock is released.

  // Return.
  return OK;
```

E. Stop Policy

A critical problem for the collaborative port scanners is to determine when to stop the scanning activities. Optimally, the collaborative port scanners stop after all hosts has been scanned for every possible vulnerability. In practice, this mission is difficult to accomplish because each host needs to make its own stop decision based on its knowledge of the global scanning activities.

Ref. [15] proposes an autonomous design. In their design, each host employs a Sum-Count-X method to determine when to stop, and communication among hosts is necessary to improve the precision of stop estimation. Ref. [16] proposes a quorum-sensing design. However, they did not consider the network topology. Also, after the stopping of the worm propagation, a worm user needs to manually restart it.

In our approach, The DHT records all scan statuses. We can constantly monitor the uninfected nodes as long as there are empty entries in the collaboration table. Therefore, the stop condition for the collaborative port scanners can be defined as all (IP address, port number) pairs have been scanned, as reflected in the DHT. We believe the collaborative-table approach is faster and more efficient since the DHT serves as the monitor and recorder of all status messages. No approximate calculation is used. If not all hosts or ports need to be scanned, users can relax the definition of the stop policy. E.g., collaborative port scanners can be defined to stop after 90 percent of all hosts are scanned.

F. Target Selection and Revisit Policy

A key step in the DHT-based collaborative port scanning is to generate random IP addresses and port numbers. Random number generators can be exploited to analyze the bandwidth of worm senders [36]. Hence, one enhancement to the scheme is to employ a variety of random number generating methods. By varying the way to generate random IP addresses and port numbers, the collaborative port scanning becomes polymorphic, and is much more resistant to analysis and defense.

To improve the efficiency of the collaborative port scanners, learning algorithms can be employed. The DHT stores a lot of information, which can be analyzed to improve future scanning activities. Moreover, the collaborative port scanners can scan a portion of all IP addresses and port numbers to reduce the scanning time. For example, they can selectively scan only

one IP in a Class C subnet, and use the scanning results to infer information about other target hosts that reside in the same subnet. Some IP addresses are employed by honeynets. To prevent collaborative port scanners from being detected and analyzed by such honeynets, the corresponding entries in the DHT for these addresses can be marked as "do not scan".

Host configurations and vulnerability statuses change over time. To capture the changes, collaborative port scanners should revisit the hosts. Some hosts have dynamic IP addresses. In such cases, collaborative port scanners can periodically revisit them and update the information on the hosts and IP addresses. A simple revisit policy is the Age Policy. In the Age Policy, there is an age attribute for each DHT entry. The age is increased by the DHT automatically and checked against a threshold. If the age reaches the threshold, the system purges its entry from the DHT to initiate a new scan. Note that researchers have studied revisit policies for web crawlers [42] and such policies could be adapted.

The DHT-based scheme needs to handle errors of participating nodes. E.g., participating nodes may provide incorrect scanning results. A simple solution to this problem is to set the revisit policy to allow each target to be scanned twice within each scanning period. Then the system could compare the results to decide if the results are usable.

G. *Comparisons and Caveats*

The DHT-based collaborative port scanning scheme can scan multiple vulnerabilities on multiple ports. In contrast, Botnets and the Curious Yellow worm [20] typically propagate by exploiting a known vulnerability on a certain port. In practice, if the target hosts do not have such vulnerability or have applied patches to fix it, the propagation will fail.

Some real-world worms, e.g., the Witty Worm [11] can check for multiple vulnerabilities. In the DHT-based scheme, attackers share knowledge about the progress and information with each other. Therefore, attackers can check for a large number of vulnerabilities and choose one to exploit.

While the extended hit-list methods consist of information on IP addresses, the DHT-based scheme records information on not only IP addresses, but also attributes for each host or subnet. Example attributes include whether the hosts are web, mail, or DNS servers, a ranked list of existing vulnerabilities, and the name and version of the host operating system. During the collaborative port scanning information regarding the configuration of victim hosts can be fingerprinted to facilitate the launch of future attacks.

VI. EXPERIMENTS

In this section, we present the experimental evaluation of the DHT-based collaborative port scanning scheme. We conduct experiments to verify our theoretical analysis, in particular: the impact of the DHT-based scanning scheme.

A. *Experiment Setup*

The experimental network consists of Intel Dual Core workstations and virtual machines running Windows XP. Without loss of generality, we use IPv4 networks and set the size of target IP address space to 2^{32}. The scanning methods used in the experiment include TCP scanning, UDP scanning, and version detection [1] that could return the system and version information running on the target hosts. We ran port scans on the target hosts to understand the latencies associated with scanning. Such latency information are used to simulate port scanning. OpenDHT [45] is also used to understand the latencies with DHT systems. We employ the hybrid architecture discussed in Section V-B.

B. *Experiments on the performance of the DHT-based collaborative scanning scheme*

We conduct experiments to study the performance of the proposed collaborative port scanning scheme. In our experiment, there are 1,000 target hosts that need to be scanned. We compare the performance of 4 different scanning setups:

1) 10 collaborative port scanners. In this setup, there are 10 collaborative port scanners that employ the fast intelligent DHT-based collaborative scanning scheme. The 10 collaborative port scanners divide the 1,000 target hosts into 2 groups. Each group has 500 target hosts. They conduct the scanning group by group, i.e., they only start to scan the second group of targets after finishing scanning the first group of targets. (We discuss more on the collaboration methods in the next experiment.) The collaborative scanners keep each other informed of the progress of the whole scanning through the DHT. The algorithms employed by the 10 collaborative port scanners are discussed in Section V.
2) 10 port scanners that operate with the static division scheme. In this setup, the 10 port scanners are divided into 2 groups, and each group has 5 port scanners. The two groups of scanners operate individually. Within a group, the 5 port scanners divide the targets statically into 5 parts, and each of them will be responsible for approximately one fifth of the target hosts.
3) 10 port scanners that operate individually. In this setup, there are 10 port scanners, but they just scan randomly without communicating with each other. As soon as the port scanner finishes scanning 50 hosts (one twentieth of all target hosts), it reports the results to a central node, and waits for the signal from the central node. The central node collects the scanned results from all scanner nodes, and combines their results. As soon as the central node finishes combining scanning results from all 10 scanners, it sends signals to all 10 port scanner nodes. In the next round, each port scanner only scans targets that have not been scanned in previous rounds. This procedure is repeated until all hosts are scanned.
4) a single port scanner. In this setup, there is only one port scanner. It is responsible for scanning all the target hosts.

All port scanners scan the well known ports for each victim host in this experiment. We impose a limit of 20 on the number of connections which a single port scanner can initiate to a target host. A typical DHT lookup takes approximately 3

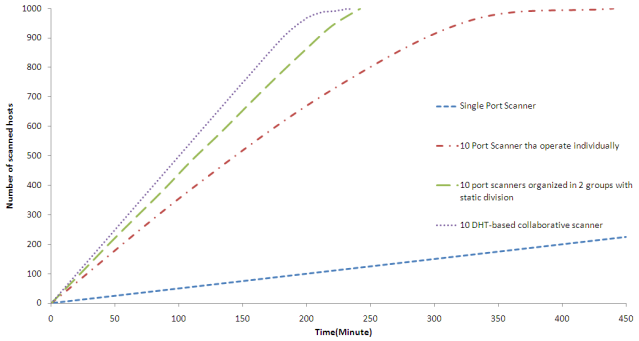

Fig. 5. The performance of the DHT-based collaborative scanning scheme.

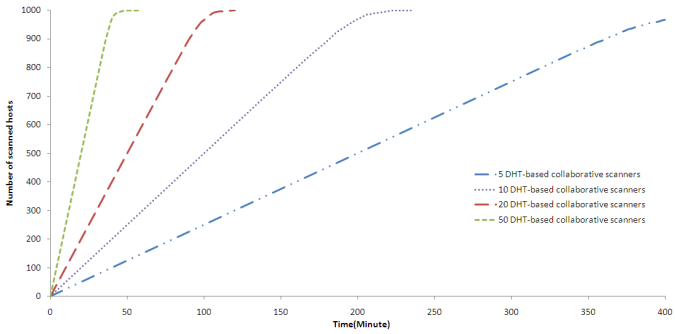

Fig. 6. The performance of collaborative scanners with different participants.

seconds with 10 active scanners. Note that a scanner has to write the results to DHT after a successful scan. A typical port scanning for one victim host that covers TCP ports and version information takes 2 minutes.

Fig. 5 shows the number of successfully scanned hosts over time for the 4 scanning setups.

We observe that, for the DHT-based 10 collaborative port scanner setup, on average, the time of the work spent in the core port scanning part constitutes most of the total operation time. It takes the collaborative port scanners 216 minutes to scan all the target hosts. The overhead ratio of the collaboration, including storing and retrieving the scanning status for the target hosts, is approximately 8 percent.

Our results show that the DHT-based 10 collaborative port scanners clearly outperform the other 3 non-DHT-based setups, and that the performance of the single port scanner is the worst among all setups. The explanation is that the DHT-based collaborative port scanners are able to perform port scanning concurrently with much more resources and minimal contention.

The experimental results verify our analysis and confirm the performance of DHT-based collaborative port scanners.

C. Experiment on the Number of Participating Collaborative Scanners

In our first experiment, 10 DHT-based collaborative port scanners collaborate with each other to conduct port scanning. Questions then arise as how would the performance of the collaborative port scanners change, as the number of participating nodes change. One would expect that a larger number of participating nodes increase the number of scanned target hosts within a specific time.

However, more scanner nodes could generate more network traffic and impose larger overhead on the DHT due to a large number of scanning status lookup and store requests. In the extreme case, an infinite number of participating nodes generate excessive traffic and overburden the DHT, effectively rendering a Distributed Denial-of-Service (DDoS) attack. Therefore, we cannot arbitrarily increase the number of scanner nodes.

We conduct experiments to find out the relation between the collaborative port scanners and the number of participating scanner nodes. Note that different DHT systems and scanning methods have different latencies and could affect such relation. Hence, in our experiments, we vary the scanning methods and use different gateways of the DHT system to create different latencies.

More thorough scans and farther gateways typically have higher latencies. If scanning latency is extremely low when compared to DHT latency, a small number of participating scanner nodes that employ the static division scheme would perform well. The reason is that a large number of nodes overburden the DHT, increase DHT latency, and slow down the whole port scan.

When scanning latency is extremely high when compared to DHT latency, a large number of participating scanner nodes perform better. The explanation is that a smaller number of nodes neither have enough parallelism nor fully utilize the DHT system. The most interesting scenario is when neither scanning nor DHT latency is too high or too low . In such cases, a large number of participating scanner nodes incur overhead on the DHT system, but could also overcome the slowness of the scanning itself.

Fig. 6 shows the number of successfully scanned hosts over time for different number of participating scanner nodes. We observe that the performance of DHT-based collaborative port scanners increase as the number of participating scanner nodes increases. However, the efficiency of the DHT-based collaborative port scanner scheme decreases as the number of participating scanner nodes increases, which confirms our analysis above.

If the number of participating scanner nodes is approximately the ratio of scanning latency to DHT latency, a good balance between the costs for maintaining a large number of scanner nodes and the performance of the collaborative port scanning can be struck.

Our theoretical analysis is as follows. Denote the number of target hosts as M, the scanning latency as S, the average DHT latency as D, the actual DHT latency as d, the number of collaborative port scanners as n, the ratio of the scanning latency to the DHT latency as k, we have:

$$k = \frac{S}{D}$$

The actual DHT latency increases as the number of collaborative port scanners increases. We assume that the increase is linear, hence:

$$d = D * n$$

The total latency L for one parallel scan is the sum of the scanning latency and the DHT latency :

$$L = S + d$$

The number of parallel scans needed for M hosts P is

$$k = \frac{M}{n}$$

Hence, the total scanning time

$$\begin{aligned} T &= P * L \\ &= (S + D * n) * \frac{M}{n} \\ &= (D * k + D * n) * \frac{M}{n} \\ &= M * D * (1 + \frac{k}{n}) \end{aligned} \tag{1}$$

Note that n should be no more than M. When n = M, the total scanning time reaches its minimum at M * D.

In the real world, we may not be able to include M scanners. However, when n is equal to k, the total scanning time is simplified to 2 * M * D, which is at least half as fast as the fastest possible scanning.

In our experiments, the scanning and DHT latencies are 2 minutes and 6 seconds (on average with 10 collaborative port scanners), respectively. Hence, the optimal number of scanner nodes is

$\frac{120}{6}$ = 20.

The experimental results confirm our analysis. With 10 scanner nodes, the overhead of the DHT-based collaborative scanning scheme is approximately 8 percent. With 20 scanners nodes, the overhead is approximately 10 percent. With 50 scanner nodes, the overhead is approximately 30 percent. Note that the quality of the DHT system affects the overhead with respect to different number of scanner nodes. In the real world, if more efficient systems can be utilized, the number of collaborative port scanners can be very large and still does not incur too much overhead. We discuss this issue more in Section VI-E.

D. Experiments on the impact of revisit policy

As discussed in Section V, collaborative port scanners can implement revisit policies. Questions then arise as whether the revisit policy would have a large impact on the performance of collaborative port scanners and how to devise a revisit policy. In this experiment, we quantitatively measure the impact of revisit policy on the performance of collaborative port scanners. There are 20 collaborative port scanners and 1,000 target hosts in this experiment. The 20 scanners are divided into 2 groups that collaborate through the DHT. Scanning of one target host takes 2 minutes to finish. For the revisit policy, making a target host revisit-able is implemented as removing its scanning status entry in the DHT. Our experimental revisit policies include:

1) Make the target hosts revisit-able after 50 minutes.
2) Make the target hosts revisit-able after 100 minutes.
3) Make the target hosts revisit-able after 150 minutes.
4) Make the target hosts revisit-able after 500 minutes.

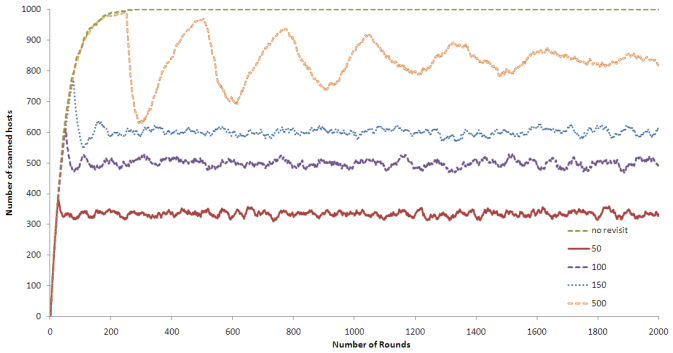

Fig. 7. The performance of the scanning with different revisit policies.

Fig. 7 shows the numbers of scanned hosts with different revisit policies. The no revisit line depicts the regular collaborative scanning scheme that does not implement a revisit policy.

We observe that revisit policy has a large impact on the performance of the collaborative port scanners. Specifically, the revisit time significantly affects the number of scanned hosts over time. We observe that as the collaborative scan progresses, the revisit policy kicks in at a certain point, depending on the pre-set revisit time.

Since a lot of hosts scanned in the beginning can ”expire” according to the revisit time, the revisit policy causes a reduction in the number of hosts that the system considers as ”scanned”. The number of scanned hosts then fluctuates as the collaborative port scanners scan the expired hosts and other hosts become expired.

Although the number of scanned hosts is not constant as the number of rounds increases, we observe that the system reaches an equilibrium around a certain number of hosts. As the revisit time increases, the equilibrium number increases as well.

However, there is a tradeoff. The revisit time cannot be arbitrarily increased because longer revisit time means that the scanning results for the target hosts are less up-to-date. The revisit time cannot be arbitrarily decreased either. If the revisit time is set as the time required to perform scanning on one host, as soon as the scanning of one target host completes, the scanning results for another target host could expire and arbitrarily delay the whole port scanning. Note if the revisit time is approximately twice as much as the time required to scan all victim hosts, the system will scan all target hosts again, which renders low efficiency.

We infer some guidelines for setting up the revisit policy. If the Age Policy discussed in Section V-F is employed, the scanning latency for one target host takes time t, the size of target host space is h, the number of scanners is s, it is recommended that the system makes the target hosts revisit-able after at least

$\frac{ht}{s}$ time.

In our experiment, the recommended revisit time is

$\frac{1000*2}{20}$ = 100 (minutes),
and the system is able to maintain scanning results for approximately half of all hosts. Note the Age Policy might not be the best policy. As discussed in section V-F, more complex revisit policies can be employed to improve the efficiency of the system.

E. Discussions on Deployment and Defense

In our experiments, we have examined the performance of the DHT-based scanning system. Large-scale deployment of the DHT-based collaborative attack scheme in the real world needs attention on a number of issues. Defenders of collaborative port scanners can mitigate the attacks by employing countermeasures to these issues.

Detection Avoidance. Certain routers and firewalls check network traffic against pre-set thresholds. Defense mechanisms such as the communication pattern analysis [19] could reveal the centralized servers. The DHT-based collaborative attack scheme could employ several methods to avoid detection: 1) limit the rate for sending and receiving packets; 2) create decentralized traffic that obfuscate the communication pattern analysis; and 3) employ encryption for the DHT-related packets whenever possible, since most existing defense systems cannot analyze encrypted traffic.

Integrity of the DHT. DHT functionality is crucial in the collaborative port scanning scheme. If the defenders can locate the DHT servers, they could try to disconnect, mislead, and defend against them by offense [9]. Hence, protecting the DHT against misuse, errors, and attacks is very important for the real-world collaborative scanners. Mechanisms that can help protect the DHT include user authentication and encryption of the stored IP address, port number, and scanning status values. User authentication may introduce extra communication latency and hurt the performance of the DHT-based collaborative scanning scheme, while the encryption of the information stored in DHT may increase the time to store and look up scanning statuses.

Scale of the DHT. In practice, the collaborative port scanning system may implement its own fast DHT system. We have seen similar large-scale system in industry, including BigTable [2] of Google and Dynamo [3] of Amazon. The DHT system can run on a large number of attacker nodes in the hybrid mode, e.g., 10,000, in practice. If more attacker machines are allocated to the DHT system, however, there will be fewer hosts responsible for the actual scanning, which can result in deteriorated scanning performance. If the collaborative port scanning system can utilize some publicly-available infrastructure and leverage its power, the overall scanning performance could improve significantly.

VII. Conclusion

In this paper, we study the collaborative port scanning, in which attackers collaborate to search the network for open ports that could be exposed to attacks. We propose different collaborative scanning strategies and analyze their advantages and disadvantages. We discuss the static, dynamic, and hybrid allocation schemes and how to employ DHT in the system. We conduct experiments to evaluate the performance and overhead of the collaborative port scanning system.

Our results show that DHT-based collaborative port scanning is a promising approach. It provides good performance, and proves that attacks can be launched by collaboration. As network speed increases in the future, we may witness an increased number of sophisticated collaborative attacks that orchestrate the computing power of a large number of attackers.

Our results suggest that issues like the number of collaborative attackers in the system, different methods of collaboration, and revisit policy all significantly affect the performance of the collaborative port scanners. We discuss issues that need consideration in real-world deployment and defense mechanisms that could mitigate the collaborative port scanners.

There are a number of ways to enhance or defend the collaborative scanning scheme, which are the subjects for future work.

First, the port scanners can employ the insider collaboration technique and attack from multiple strategic locations. In the insider collaboration attack, an insider gathers the knowledge about vulnerable hosts. The outsider launches port scanning with the pre-acquired knowledge from the insider. In this case, the knowledge of vulnerable hosts can be gathered offline rather than online. Another point worth mentioning is, outside attacks can easily be blocked by firewall, while insider attacks normally can bypass firewall checks.

Second, port scanners can also employ heuristics and more intelligent algorithms. For example, learning algorithms can be utilized. However, such port scanners may scan very slowly due to the complicity of the algorithms. Solutions include offline training of the scanner with a lot of log data. Smarter revisit policies can be employed as well. Handling dynamic IP address change and improving the efficiency of the scanners are of interest as well.

Third, collaborative port scanners can employ the passive and stealth technique. For example, they can passively log and analyze the network traffic. Such techniques could enhance the collaborative port scanners and challenges the defense systems.

Fourth, we could fingerprint the collaboration methods employed by the collaborative attackers. Defense systems can implement algorithms that learn and classify the communication patterns like the flooding, server-based, and distributed architectures. Then, they can monitor and analyze the network traffic to detect these collaboration patterns and flag corresponding nodes as possible collaborative attackers.

Finally, addressing the real-world issues and implementing the DHT-based scanner idea as a large-scale system are the most interesting piece of future work.

Acknowledgment

The authors would like to thank Dr. Yi Mao for helpful discussions.

References

[1] http://nmap.org/book/man-performance.html

[2] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A Distributed Storage System for Structured Data. In Proc. of the 7th Symposium on Operating System Design and Implementation, 2006.

[3] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: amazon's highly available key-value store, Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, October, 2007

[4] Burton H. Bloom, Space/Time Trade-offs in Hash Coding with Allowable Errors, Communications of the ACM, Vol.13, 1970

[5] Y. Zhang and B. Bhargava, The Effects of Threading, Infection Time, and Multiple-Attacker Collaboration on Malware Propagation, The 28th IEEE International Symposium on Reliable Distributed Systems (SRDS 2009), September, 2009. Niagara Falls, New York, U.S.A

[6] S. Sarat, A. Terzis, Measuring the Storm Worm Network. Technical Report 01-10-2007, http://hinrg.cs.jhu.edu/uploads/Main/STORMTR.pdf

[7] C.Kanich, K.Levchenko, B.Enright, G.M.Voelker and S.Savage, The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff, Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), San Franciso, CA, April 2008

[8] Kademlia Specification http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html

[9] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, DDoS Defense by Offense, ACM SIGCOMM 2006, Pisa, Italy, September 2006

[10] Z. Chen, L. Gao, and K. Kwiat, Modeling the Spread of Active Worms, IEEE INFOCOM 2003

[11] http://www.caida.org/research/security/witty/, last accessed Apr 20, 2008

[12] J. Yang. "Fast Worm Propagation in IPv6 Networks" http://www.cs.virginia.edu/ jy8y/publications/cs85104.pdf

[13] http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

[14] S. Staniford, V. Paxson and N. Weaver. "How to Own the Internet in Your Spare Time" In Proceedings of the 11th USENIX Security Symposium, August 2002

[15] J. Ma, G. Voelker and S. Savage, Self-stopping Worms, Proceedings of the ACM Workshop on Rapid Malcode (WORM), Washington D.C., November 2005.

[16] R. Vogt, J. Aycock, and M. Jacobson, Jr. Quorum Sensing and Self-Stopping Worms. Proceedings of the 5th ACM Workshop on Recurring Malcode (WORM 2007), Alexandria, VA, November,2007.

[17] Detecting and Recovering from a Virus Incident http://www.sans.org/reading_room/whitepapers/malicious/903.php

[18] D. Dagon, G. Gu, C. Lee, and W. Lee. "A Taxonomy of Botnet Structures." In Proceedings of the 23 Annual Computer Security Applications Conference (ACSAC'07), Miami Beach, FL, December 2007.

[19] G. Gu, J. Zhang, and W. Lee. "BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic." In Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08), San Diego, CA, February 2008

[20] B. Wiley, Curious Yellow: The First Coordinated Worm Design, http://blanu.net/curious_yellow.html, Accessed Apr 20, 2008

[21] Z. Chen and C. Ji, A Self-Learning Worm Using Importance Scanning, ACM CCS Workshop on Rapid Malcode (WORM05), 2005

[22] C. Zou, D. Towsley, and W. Gong. "On the Performance of Internet Worm Scanning Strategies," Elsevier Journal of Performance Evaluation, July 2006

[23] Z. Chen and C. Ji, Optimal Worm-Scanning Method Using Vulnerable-Host Distributions International Journal of Security and Networks: Special Issue on Computer and Network Security, vol. 2, 2007

[24] J. Wu, S. Vangala, L. Gao, and K. Kwiat, An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques, Network and Distributed System Security Symposium 2004

[25] J. Twycoss, M. Williamson: Implementing and Testing a Virus Throttle. In: Proceedings. 12th USENIX Security Symposium, Washington, 2003

[26] M. Vivo, E. Carrasco, G. Isern, G. Vivo, A review of port scanning techniques, ACM Computer Communications Review,Volume 29, Apr. 1999

[27] A. Voyiatzis, D. Serpanos: Pulse: A Class of Super-Worms against Network Infrastructure. ICDCS Workshops 2003

[28] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of ip address lookup algorithms," IEEE Network Magazine,vol.15,Mar.-Apr. 2001

[29] J. Jung, V. Paxson, A. Berger, and J.Balakrishnan, Fast Portscan Detection Using Sequential Hypothesis Testing, In Proc. of the IEEE Symposium on Security and Privacy, May 2004

[30] S. Staniford, J. Hoagland, J. McAlerney: Practical Automated Detection of Stealthy Portscans. Journal of Computer Security 10(1/2), 2002

[31] S. Bellovin, B. Cheswick, A. Keromytis. Worm propagation strategies in an IPv6 Internet. http://www.cs.columbia.edu/ smb/papers/v6worms.pdf, LOGIN, Vol 31. No.1.

[32] P. Wang, S. Sparks, C. Zou. "An Advanced Hybrid Peer-to-Peer Botnet", preprint, IEEE Transactions on Dependable and Secure Computing, 2009

[33] A. Wagner, T. Dubendorfer, B. Plattner, R. Hiestand, Experiences with Worm Propagation Simulations ACM Workshop on Rapid Malcode (WORM), 2003

[34] M. Vojnovic, V.Gupta, T.Karagiannis, and C.Gkantsidis, Sampling Strategies for Epidemic-Style Information Dissemination, IEEE Infocom, 2008

[35] A. Kamra, H. Feng, V. Misra and A. Keromytis, The Effect of DNS Delays on Worm Propagation in an IPv6 Internet, Proceedings of IEEE Infocom, IEEE, Miami, FL, USA, 2005.

[36] A. Kumar, V. Paxson, N. Weaver, Exploiting Underlying Structure for Detailed Reconstruction of an Internet-scale Event. In the proceedings of ACM IMC, New Orleans, LA, Oct 2005.

[37] C. Gates, Co-ordinated Port Scans: A Model, A Detector and An Evaluation Methodology. PhD Thesis. Dalhousie University. Feb., 2006

[38] S. Friedl, Analysis of the new "Code Red II" Variant, http://www.unixwiz.net/techtips/CodeRedII.html, Last Accesses Apr 15, 2008

[39] C. Zou, W. Gong, D. Towsley. "Code Red Worm Propagation Modeling and Analysis," 9th ACM Conference on Computer and Communication Security (CCS'02), Nov. 18-22, Washington DC, USA, 2002

[40] D. Moore, C. Shannon, and J. Brown. Code-Red: a case study on the spread and victims of an Internet Worm. In Proc. ACM/USENIX Internet Measurement Workshop, France, November, 2002

[41] H. Balakrishnan, M.Kaashoek, D.Karger, R.Morris, and I.Stoica. Looking up data in P2P systems. In Communications of the ACM, February 2003.

[42] J. Cho, H. Garcia-Molina "Effective page refresh policies for Web crawlers." ACM Transactions on Database Systems, 28(4): December 2003.

[43] http://www.bittorrent.com/

[44] http://www.emule-project.net/

[45] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A Public DHT Service and Its Uses. Proceedings of ACM SIGCOMM 2005, August 2005.

[46] Strange Attractors and TCP/IP Sequence Number Analysis - One Year Later, http://www.iu.hio.no/ haugerud/ids/SAATSNA_OYL.pdf

[47] http://www.distributed.net/