Hindawi
Wireless Communications and Mobile Computing
Volume 2021, Article ID 6627639, 10 pages
https://doi.org/10.1155/2021/6627639

Research Article

A New Approach Customizable Distributed Network Service Discovery System

Xiangzhan Yu, Zhichao Hu, and Yi Xin

School of Cyberspace Science, Harbin Institute of Technology, 150001, China

Correspondence should be addressed to Xiangzhan Yu; yxz@hit.edu.cn

Received 31 December 2020; Revised 15 March 2021; Accepted 23 April 2021; Published 12 May 2021

Academic Editor: Lihua Yin

Computer systems and applications on the internet provide services to outsiders and, at the same time, the vulnerabilities may be exploited by attackers and leak some sensitive private information. To collect and monitor the service information provided by the network environment such as IoT (Internet of Things), vehicular networks, cloud computing, and cloud storage, it is particularly important that a system can provide faster service discovery for discovering and identifying specific network services. The current service discovery systems mainly use port scanning technology, including Nmap, Zmap, and Masscan. However, these technologies hard code the service features and only support common services so that cannot cope with real-time updates and changing network services. To solve the above problems, this paper proposed a customizable distributed network service discovery system based on stateless scanning technology of Masscan and proposed a customizable interactive pattern set syntax. The system used random destination address technologies to scan for Ipv4 address allocation and used a distributed deployment scheme. Experimental results show that the system has high scanning speed and has high adaptability to new services and special services.

1. Introduction

With the growth of internet devices and applications, various large scale cyberattacks continue to emerge, and internet vulnerabilities also show a surging trend [1, 2]. Despite the recent growth in computer networking best practices, the continual improvement in Internet-based services has presented new challenges in maintaining security and preserving privacy [3, 4]. Even though some enterprises have discovered vulnerabilities and released repair patches, many users still do not update, leading to potential security threats and providing attackers with access to attacks. At the same time, many web apps and services are installed on the devices hosting a web client and providing the interface for user control with open ports, where security and privacy are the critical issues [5, 6]. Censorship needs to know these security risks, that is, to count and supervise the service information in a large-scale network.

The IoT, vehicular networks, cloud computing, cloud storage, and other environments can provide users with flexible and convenient service access [7, 8]. While greatly improving the convenience of life, privacy issues caused by security problems are also becoming more and more serious [9, 10]. For example, in vehicular networks, security plays a dominant role as applications based on vehicular networks usually correspond to passengers' safety (e.g., self-driving) and privacy information (e.g., driving history) [11]. So the security of the network should be one of the most important issues in the upcoming days. Searching and gathering the specific information of the devices on the internet provide data to analyze the vulnerabilities which can enhance system's security and preserve privacy [4, 12]. A common tool to deal with this problem is port scanning, but current scanning tools have two disadvantages. In one hand, supported services are mostly hard coded in the system, and for less common, newer services, you need to wait for the developer's update support. It has poor scalability, as evidenced by the famous Masscan, which only supports HTTP, SSL, and other common protocols but ignores industrial network protocols and instant messaging protocols. On the other hand, the traditional scanning methods are noninteractive detection, so they are failed for service identification with multiple interactions.

In order to solve the above problems, we designed a customizable distributed network service discovery system (CDNSDS) in this paper. The main contributions of this work are as follows:

(i) We designed a system architecture, which includes three subsystems: central control subsystem (CCS), schedule control subsystem (SCS), and scanning proxy subsystem (SPS). The CCS is the brain; it receives the user's instructions and manages and assigns tasks to the SCS. The SCS is the bridge connecting CCS and multiple SPS. The last subsystem is the SPS, the key factor for performance. We optimized Masscan, the most efficient scanning tool currently, and used a distributed program to improve concurrency

(ii) To be customizable, we had compiled a pattern set of syntax conventions. The syntax conventions can convert the user's customized services description, including interactive service, to standard syntax which is accepted by a scanning tool in the SPS

The rest of the paper is organized as follows: a related work is described in Sections 2 and 3 elaborates the proposed system CDNSDS; experimental results are followed in Section 4. Finally, we summarize the research in Section 5 with a discussion as well as a future work.

2. Related Work

In the study of empirical security, fast Internet-scale network service discovery has opened a new avenue, while scanning technology plays an important role. One of the earlier scanning tools is Nmap [13], which maintains a full-connected state to track hosts that have been scanned and to handle timeouts and retransmissions. In this state, the unresponsive requests cost too much time; it takes several weeks or many machines for Nmap to scan the public address space. To overcome the issue of efficiency, Zakir et al. [14] designed a scanning tool Zmap based on no per-connected state. For Zmap, there is no need to track connection timeouts, and it accepts response packets with the correct state fields during the scanning. The manner of Zmap is similar to SYN cookies. Compared to Nmap, with the same accuracy, Zmap is capable of scanning the IPv4 public address space for under 45 minutes on a single machine [15], which is over 1300 times faster than the most aggressive Nmap default settings. Further, drawing on the data collected by Zmap from ongoing Internet-wide scanning, Zakir et al. [16] designed a public search engine named Censys, which supports full-text searches on protocol banners and querying a wide range of derived fields. With Censys, it becomes simple to help researchers answer security-related questions.

Although Zmap has greatly improved in performance, scanning technology is still in progress. The fastest internet port scanner Masscan [17], an open source project, only takes six minutes to scan the IPv4 public address space, transmitting 10 million packets per second. For the sake of high performance, Masscan takes endeavour from three aspects. For one thing, similar to Zmap is the use of no per-connected state. Because Masscan can simultaneously maintain the number of connections which is set by the program itself, the number can be set very large, so the scanning speed is much faster than other scanners. For another, Masscan uses a custom TCP/IP stack, and a designated network device and PF_RING DNA driver are necessary conditions. It is a lightweight protocol stack that means the underlying packet processing, connection control, etc. will bypass the operating system protocol stack, so the protocol stack process is simpler and there will be a substantial increase in performance. In addition, the configuration of Masscan is more flexible, not limited to single-port probing, and a user can specify the port segment. Through a target address randomization algorithm, it can be more effective to random host range for target that can evade from detecting of Intrusion Detection System (IDS).

Except for the above famous scanners, a number of research efforts focus on empirical security. In order to scan anonymously, Rodney et al. [18] performed scanning through Tor, which can hide the source's IP address from the target. Andrei et al. [19] proposed a public, large-scale analysis of firmware images, which supported a global understanding of embedded systems' security. At the same time, the Heartbleed vulnerability is the measurement and analysis in [20]. In the weak keys detecting, researchers [21–23] reported they had computed the RSA private keys for HTTPS hosts on the internet and traced the underlying issue to widespread random number generation failures on network devices. Arzhakov et al. [24] proposed a multithread network scanner with a very flexible architecture that allows us to parallelize the process of sending requests and receiving responses from remote hosts. Focused on automated web scanners, Fang et al. [25] gave a new direction for the detection of the fingerprint using a finite state machine to abstract differences of scanners.

3. Service Discovery System

3.1. System Architecture. The traditional service discovery systems may cause issues such as triggered IDS alarm and single-node detection poor performance. In this paper, we design and implement the CDNSDS and the architecture is shown in Figure 1.

CDNSDS includes three subsystems.

(i) Central control subsystem (CCS): it is the brain, which receives a user's instructions and manages and assigns tasks to the SCS. Users can get the task process and results and manage the attribute and state of the scanning node. In this subsystem, we design a pattern set of syntax conventions to support customizability.

(ii) Schedule control subsystem (SCS): it is a bridge connecting CCS and multiple SPS. It provides task division, scheduling management, and results of the temporary service.

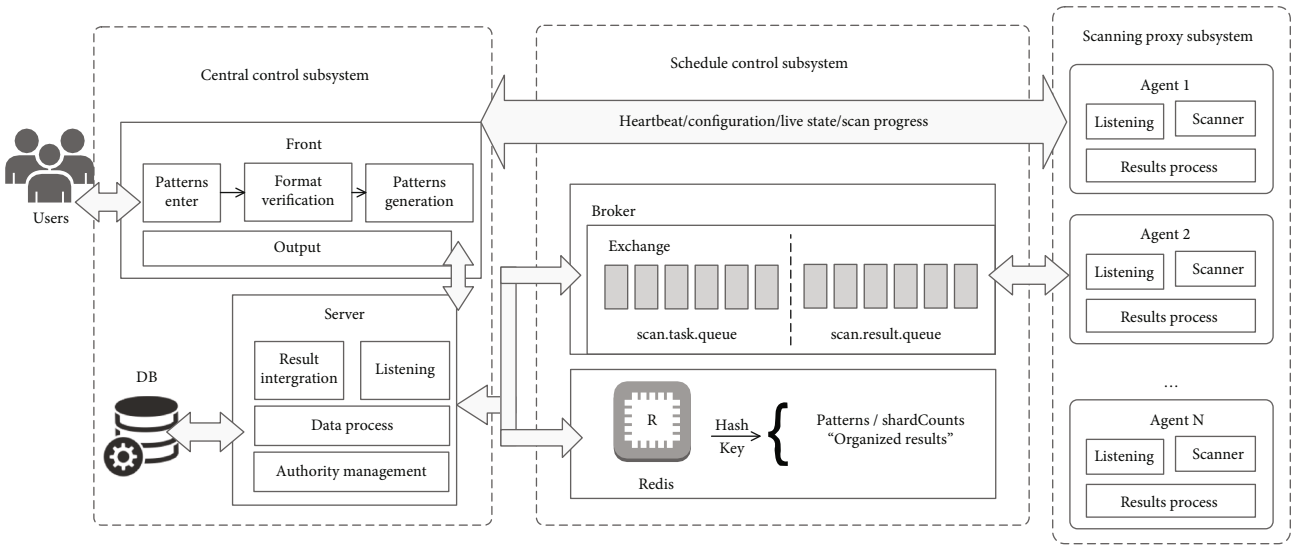

Figure 1: System architecture.

Table 1: The property of pattern set.

Name	State	Description	Format	Essential
msg	s	After the state is transferred to this node, the text in the msg attribute needs to be filled into the application layer load; the data packet is constructed and probed	Hex/string	Y
waiting	s	After sending the probe packet in this state, state will be transferred to receive state when receives a response packet	"r" + index	Y
isbanner	r	Output application layer load to output file	True/false	N
patterns	r	Probe pattern set, which is used to guide state transitions	Hex/string	N
len	r	Limit of payload length	Range	N
goto	r	The state is after filtering of patterns and len	{index, state_with_id}	Y

(iii) Scanning proxy subsystem (SPS): it is the key factor for performance. The SPS consists of several distributed agent modules. Each agent is a scan node with optimized Masscan that performs real-time scan task from the SCS.

3.2. Central Control Subsystem (CCS). The CCS provides users with customizable service probe interfaces. A pattern set of syntax conventions is defined for customization as follows.

We use s for send state and r for receive state. Denote $D = \{s_i, r_j\}$ as instructions, s_i is the ith state of send, and r_j is the jth state of receive. The attributes of different state are split with character '.'. Property set of send state s is

$$P_s = \{\text{msg}, \text{waiting}\}, \tag{1}$$

and property set of receive state r is

$$P_r = \{\text{isbanner}, \text{patterns}, \text{len}, \text{goto}\}. \tag{2}$$

Table 1 shows the list of the property descriptions of P_s and P_r. An example state transition diagram is shown in Figure 2.

In this example, there are three kinds of state node: (1) the green solid nodes $s0$, $s1$, and $s2$ are send state; (2) the hollow blue nodes $r1$, $r2$, and $r3$ are receive state without banner output that means the property isbanner equals to false; and (3) the solid nodes $r0$, $r4$, and $r5$ are receive state with banner output that means the property isbanner equals to true.

3.3. Schedule Control Subsystem (SCS). The SCS is aimed building an efficient and reliable communication environment between the CCS and SPS, while providing intermediate data storage and high-speed read service.

The SCS contains three modules: state management module, message queue module, and cache module.

(i) Status management module: to better understand the survival status and scanning progress of each scanning agent node, the SCS is logically responsible for building the communication environment between the central control system and each scanning node. At the same time, to deal with the problems such as downtime of the CCS and change of server address, the SCS also provides an interface to dynamically manage the connection configuration of the scanning nodes to ensure the normal

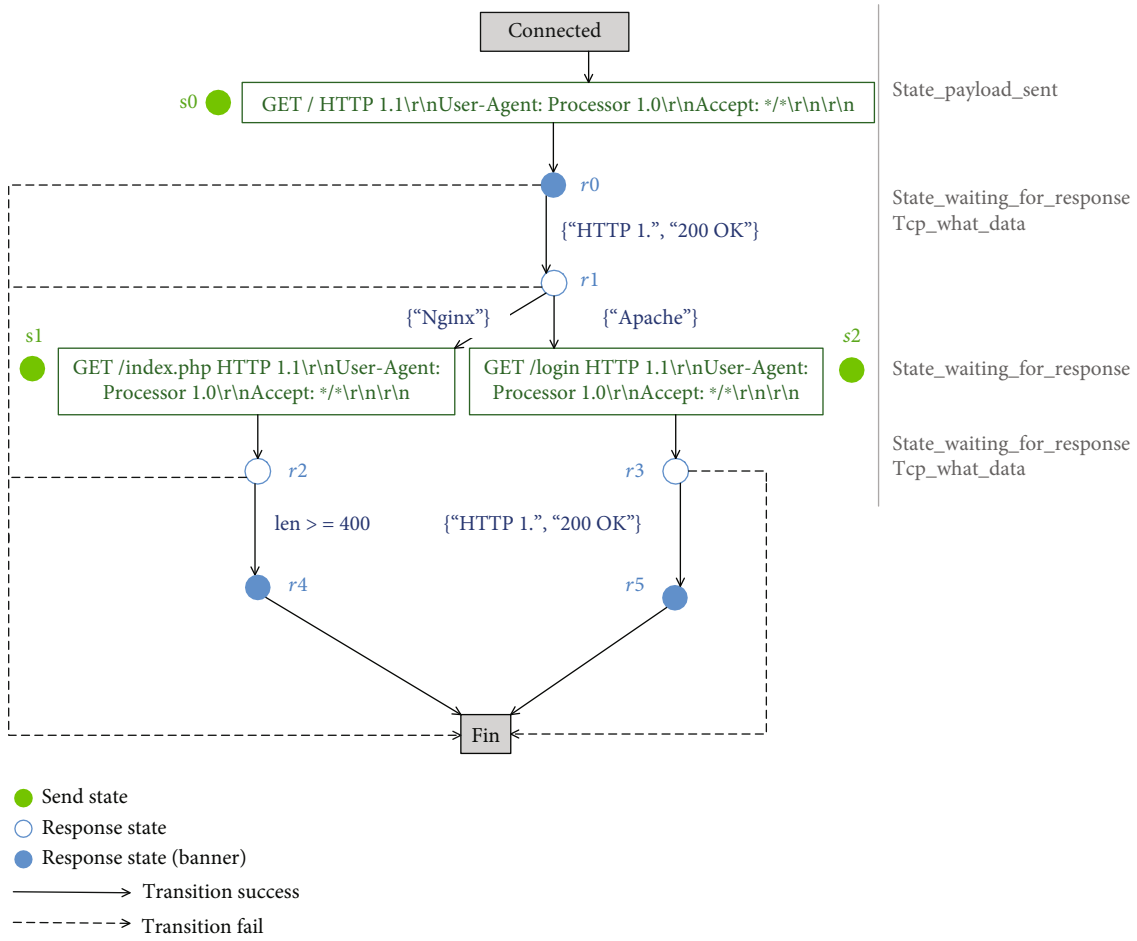

Figure 2: State transition diagram.

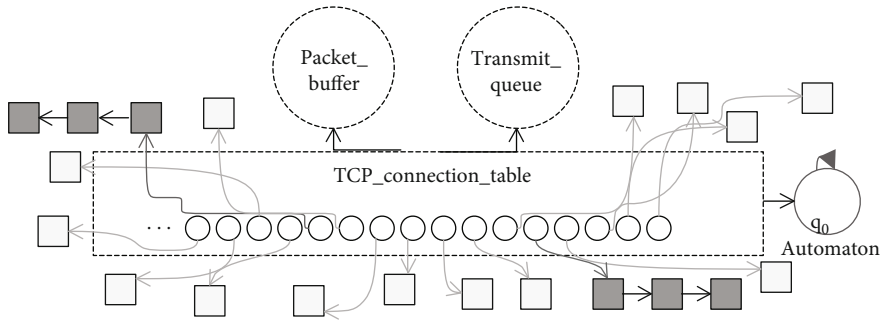

Figure 3: TCP connection table in user-mode protocol stack.

delivery of heartbeat packets and scanning progress packets.

(ii) Message queue module: it contains task queue management and result queue management. The detection tasks issued by the CCS will split into specified slices for smaller granularity and detection. Each scanning agent node consumes only one slice at a time, and these task slices will be handed over to the task queue management. From the scanning agent's point of view, each slice represents a scanning task, and the result of the task may be success, fail, or timeout. The task queue manages the various results that may exist after each task slice is received. The slice that fails to scan is reenlisted for other scanning nodes to probe again. When the task is successfully scanned, the result data will be passed back from the scanning node to the result queue, which will store that result slice temporarily for the CCS.

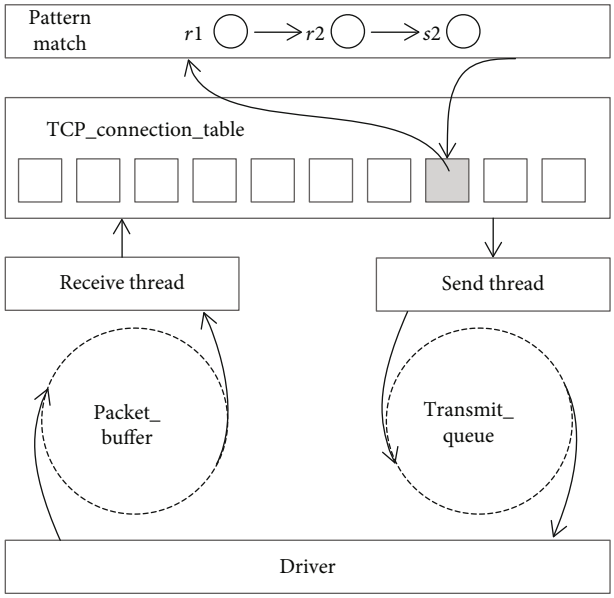

Figure 4: Interactive service detection hierarchy.

1. c ⟵ $fe[r, a, b]_K(m)$
2. If c ∈ M, Then
3. Return
4. Else
5. Return $Fe[r, a, b]_K(c)$
6. EndIf

Algorithm 1: $Fe[r, a, b]_K(m.)$

Similarly, the result queue will also manage the status of the results processed by the central control system as described above.

(iii) Cache module: after the scanning node successfully detects the target address set, it will pass the result slice back to the result queue and then open the next detection task. Due to the large number of potential detection nodes, if each slice's results are stacked in the CCS, it will increase the pressure on its storage and processing. Therefore, the module provides a dumping service to the cache and notifies the CCS to consolidate, deduplicate, and persist the results after receiving. Thus, the cache module provides memory-level high-speed data processing functions.

For each packet received, the SCS will determine whether it is a task slice, heartbeat data, or task result.

(i) Task slice packet: it is handed over to the "task queue" to manage and monitor the execution (dispatch) result of this slice.

(ii) Heartbeat packet: it is forwarded to the central control system to update the survival and progress status.

1. $L \longleftarrow m \bmod a$; $R \longleftarrow \lfloor m/a \rfloor$
2. For $j \longleftarrow i$ to r, do
3. If j is *odd*, Then
4. $tmp \longleftarrow (L + F_j(R)) \bmod a$
5. Else
6. $tmp \longleftarrow (L + F_j(R)) \bmod b$
7. EndIf
8. L ⟵ R ; $R \longleftarrow tmp$;
9. EndFor
10. If r is *odd*, then
11. Return $aR + L$
12. Else
13. Return $bR + L$
14. EndIf

Algorithm 2: $fe[r, a, b]_K(m)$

1. c ⟵ $fe[r, a, b]_K^{-1}(m)$
2. If c ∈ M, Then
3. return
4. Else
5. return $Fe[r, a, b]_K^{-1}(c)$
6. EndIf

Algorithm 3: $Fe[r, a, b]_K^{-1}(m)$

1. If (r is odd) Then
2. $R \longleftarrow mod\ a$; $L \longleftarrow \lfloor m/a \rfloor$
3. Else
4. $L \longleftarrow m \bmod a$; $R \longleftarrow \lfloor m/a \rfloor$
5. End
6. For $j \longleftarrow r$ to 1, do
7. If j is *odd*, Then
8. $tmp \longleftarrow (R - F_j(L)) \bmod a$
9. Else
10. $tmp \longleftarrow (R + F_j(L)) \bmod b$
11. EndIf
12. $R \longleftarrow L$; $L \longleftarrow tmp$;
13. EndFor
14. Return $aR + L$

Algorithm 4: $fe[r, a, b]_K^{-1}(m)$

(iii) Task slice result data: it is temporarily stored in the cache module. The task slice result data is temporarily stored in the cache module to remind the central control system for integration.

3.4. Scanning Proxy Subsystem (SPS)

3.4.1. Interactive Service Detection. When the SPS performs a scan task, there are four situations after sending the first TCP handshake request.

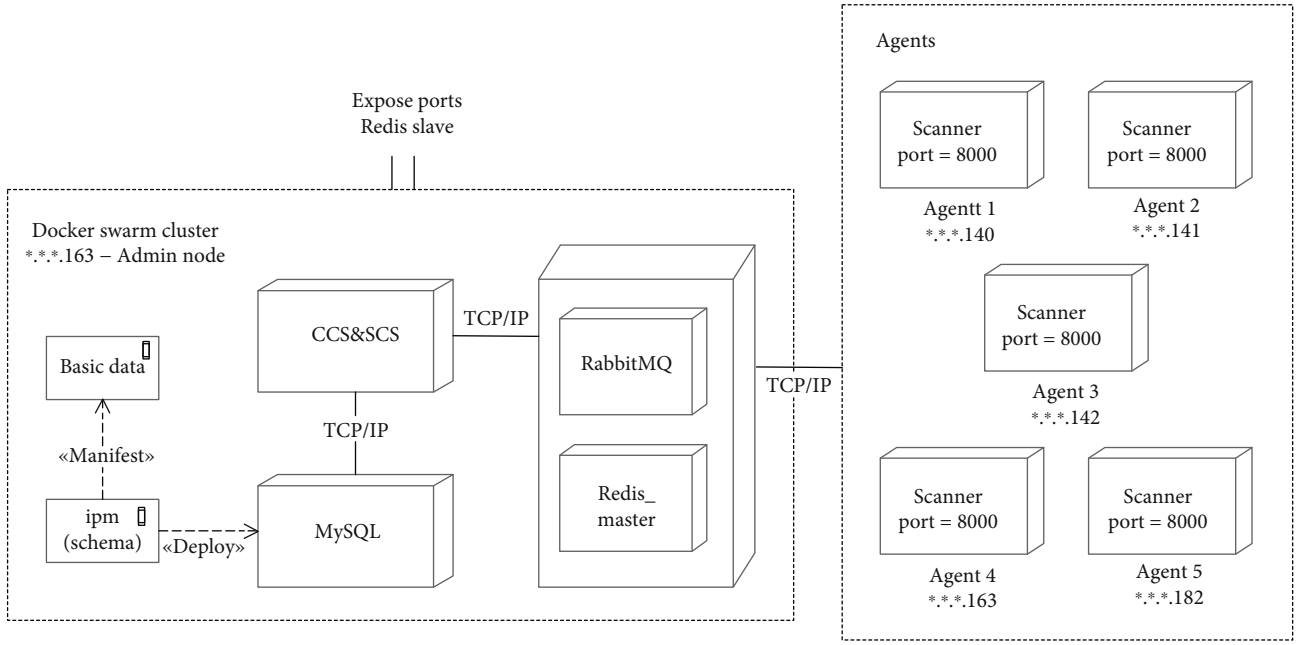

Figure 5: System deployment diagram.

(i) SYN-ACK: the host port of corresponding target is open, and we can continue to probe service.

(ii) RST: the target is open, but the destination port is close.

(iii) ICMP unreachable: the target is close.

(iv) No response: connection timed out.

It is obviously that in the first case we can keep detecting while other cases can be directly abandoned.

Normally, scanners based on semiconnected state will send RST to close connection after receiving SYN-ACK. Such a scheme is not suitable for interactive detection. This issue can be resolved by two possible solutions.

(i) Using the operating system protocol stack, reestablish the connection to the open target port for deep probing

(ii) Send ACK to finish three-way handshake instead of RST

The first solution theoretically provides reliable connection, but the number of connections is limited. The second solution requires a user-mode protocol stack and is more efficient than the former. Fortunately, Masscan already provides this functionality. Therefore, the second solution is adopted in this paper.

In order to record all active connections, a TCP connection table is needed to maintain the management of Transmission Control Block (TCB) which contains all the important information about the connection, as shown Figure 3.

The interactive service detection hierarchy is displayed in Figure 4. Through asynchronous threads, the sending and receiving are separated.

During service discovery, packets need to bypass the original system stack; otherwise, the original system stack will send RST packet because of the absence of connections. This paper proposed two solutions.

(i) ARP cheat: send an ARP packet with an unreal IP in same subnet to router.

(ii) Modify Linux iptables: drop traffic with a specified port.

3.4.2. Randomize Target Address. The CCS delivered tasks in the form of fragments. Under the premise of address randomization, in order to avoid duplication of detection intervals of all nodes, the system sets the range as $S_{\text{range}} = N_{\text{hosts}} \times N_{\text{ports}}$ to serialize the scan range.

Denote IP segment $A = \{A_1, A_2 \cdots A_n\}$, port setment $B = \{B_1, B_2 \cdots B_m\}$, IP-port consist data:

$$(A_1B_1), (A_1B_2), \cdots, (A_1B_m), (A_2B_1), \cdots, (A_nB_1), \cdots, (A_nB_m). \tag{3}$$

Scan range mapping set is follows:

$$R_1 = (A_1B_1), R_2 = (A_1B_2), \cdots, R_{n\times m} = (A_nB_m). \tag{4}$$

Using the above mapping set, the conversion from the index of the range to host addresses and ports can be achieved according to equations (5) and (6). We use this conversion to find the IP and port of the ith scan task.

$$\text{ip}_i = \text{pick}\left(\text{addresses}, \frac{i}{\text{port}_{\text{count}}}\right), \tag{5}$$

Table 2: The results of interactive service detection.

Speed (pkt/s, k: 10^3, w: 10^4)	Service	Port	Service time (s)	Port time (s)
0.1k	3078	3167	852	713
0.5k	3190	3154	178	150
1k	3239	3240	95	79
2k	3151	3157	52	46
3k	3033	3080	38	34
5k	2189	2251	26	25
1w	1218	1054	17	17

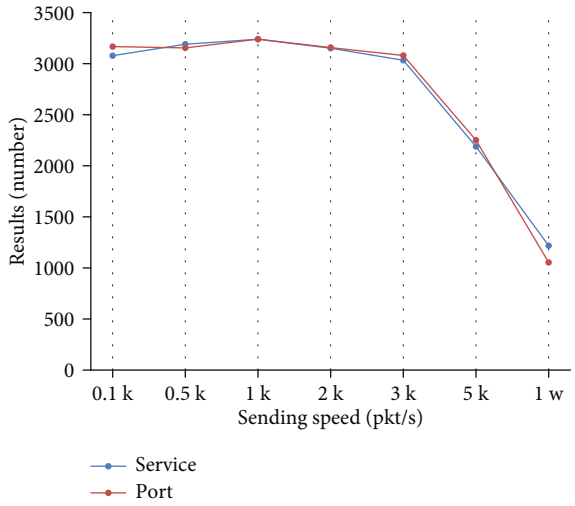

Figure 6: Interactive service detection results chart (k: 10^3, w: 10^4).

$$\text{port}_i = (\text{ports}, i\%\text{port}_{\text{count}}). \quad (6)$$

After serialization, let us suppose fragment range set is r, then randomization of r is that

$$\exists r' \in R, \quad \text{satisfying } r \neq r', \text{card}(A) = \text{card}(B). \quad (7)$$

Due to the condition r' is not unique, the degree of randomization is judged by comparing the same number of elements. The less the number, the higher the degree. In the scan module, randomize target address using generalized Feistel [26] encryption to achieve $k \times M \to M$, where k is any number and M is the target host range. In the k intervention, a mapping process to achieve the same range of random is as follows:

This method is a modification of Feistel encryption, function is Fe$[r, a, b]$, r is rounds, $a, b \in N$, and $ab \geq k$.

The process of randomize address recovery is as follows.

4. Results and Discussion

4.1. Experimental Environment and Deployment. In order to satisfy cluster operations and distributed schedule, the system adopted Docker Swarmkit and deployed in 5 nodes. Among them, in Docker Swarm Mode, the CCS and SCS

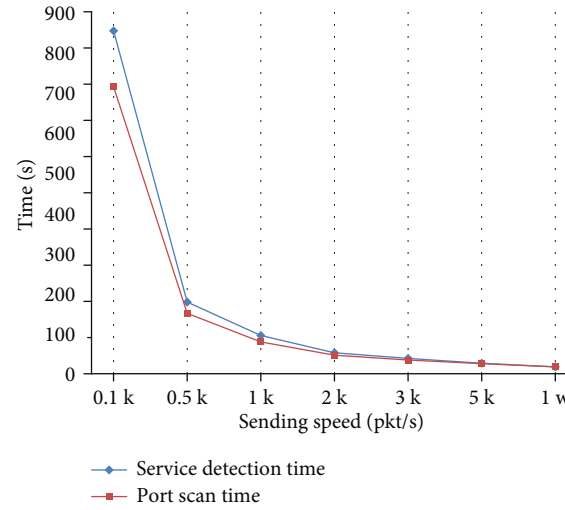

Figure 7: Interactive service detection time chart (k: 10^3, w: 10^4).

Table 3: The results of testing in multimode.

Speed (k: 10^3, w: 10^4)	Multinode: results/time (s)/target	Single-node: results/time (s)/target
0.5k	25641/1743/3	25691/8630/3
0.7k	25764/1248/3	25669/6174/3
1k	25666/877/3	25576/4334/3
2k	25709/448/3	25496/2174/3
4k	25510/230/3	25389/1094/3
6k	25246/157/3	25373/734/3
8k	25377/122/3	25520/557/3
1w	25459/101/3	25527/447/2
2w	25046/57/3	25026/230/2
3w	24968/43/3	25212/160/2
4w	24844/36/3	25137/124/3
5w	23063/31/3	25118/103/2
6w	21221/28/3	25132/88/3
7w	19018/26/2	24998/74/2
8w	16642/25/3	25277/69/3
9w	15190/24/2	25169/64/3
10w	13740/25/3	25203/59/3
20w	10590/25/1	25317/47/2

are deployed in the admin node, and another five SPS are deployed in other nodes. The system deployment diagram is shown as Figure 5.

4.2. System Testing in Single Node. In this paper, we choose a single node to probe HTTP service, the target host segment is 169.54.23.0/16, and the probe port is 80. Each set of experiments is the average of three testing. The testing results are shown as Table 2.

According to the data, we can get the following charts. As Figures 6 and 7 show, when sending rate less than 3000 pkt/s,

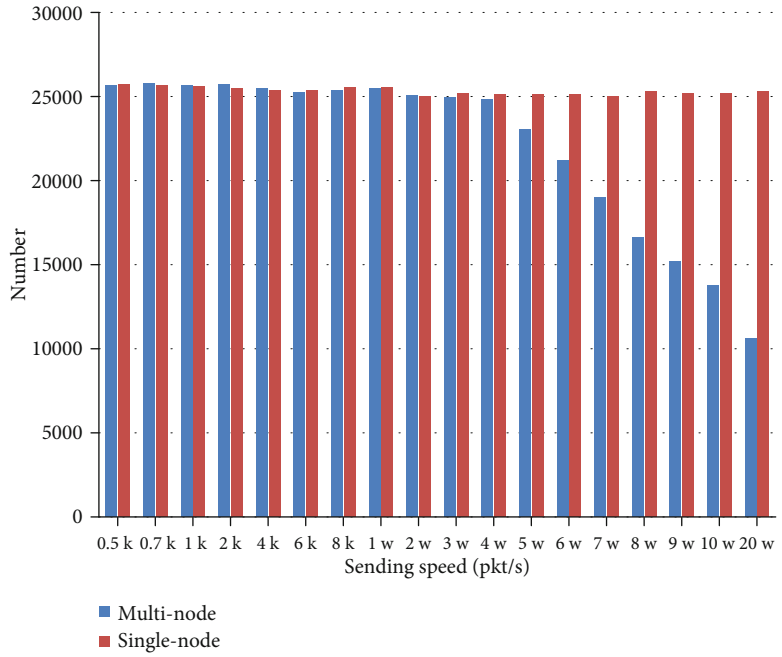

Figure 8: Fixed scanning range (k: 10^3, w: 10^4).

Table 4: The impact of scope on the result.

Scope	Multinode: results/time (s)	Single-node: results/time (s)
/24	18/13	15/13
/22	89/14	89/14
/20	199/14	200/17
/18	842/16	854/34
/16	2862/29	2854/96
/14	8271/74	8247/323
/12	13226/237	13405/1129
/10	25961/878	26281/4341
/8	40920/3427	40632/17102
/6	343482/13881	338102/69277

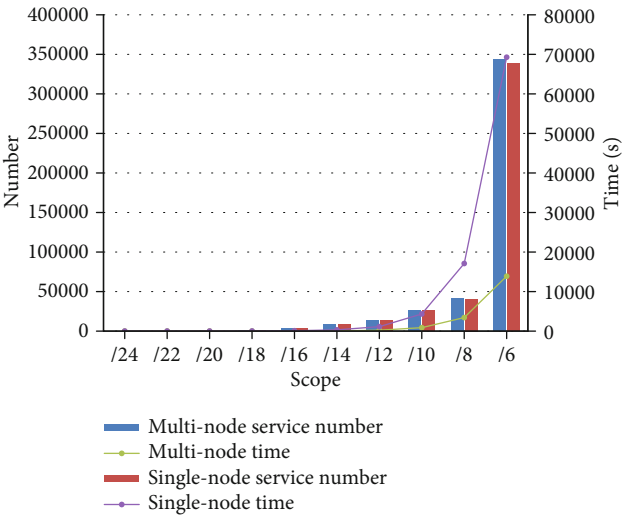

Figure 9: Fixed sending rate.

the results are generally flat, but after that, they decreased significantly. The reason is that when the rate of sending packets increases, the time reduces, so it takes time to wait for service probe packets or SYN-ACKs. Therefore, there is such a situation that the response packets arrive after the scanner shut down. 3000 pkt/s is a stable sending rate in this testing.

4.3. System Testing in Multinode. There are a total of five nodes for testing; the pattern set comes from the analysis of the protocol icoco with port 80. In order to compare the distribute platform and single node, we test in the following two aspects.

(i) Scanning range is fixed, and sending rate changes

(ii) Sending rate is fixed, and scanning range changes

For the first aspect, the target host segment is 169.54.23.0/10 and port is 80. The results are shown as Table 3, and the trend is shown in Figure 8. In Table 3, results mean the number of icoco service.

For the second aspect, the fixed sending rate is set 1000 pkt/s, the results for different scope are shown in Table 4, and the trend is shown in Figure 9.

As can be seen from Figure 9, the accuracy between multinode and single-node is similar. However, as the scanning range increases, the multinode shows better performance.

Consider ratio changes at the same sending rate of the single-node and multinode, as shown in Figure 10. In an ideal

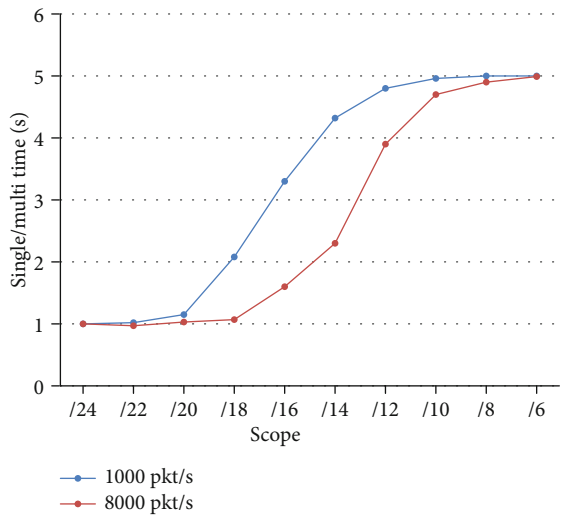

Figure 10: Single-node and multinode ratio diagram.

environment, the detection time of N nodes is $1/N$ of the single node.

As can be seen from Figure 10, reaching the ideal ratio value "5" is determined by the packet rate and the scope of the probe host. It can be derived as follows in conclusion:

(i) If the detection range is fixed, the lower the sending rate, the easier it is to approach the ideal ratio

(ii) If the sending rate is fixed, the larger the detection range, the easier it is to approach the ideal ratio

Based on this conclusion, for better detection results, the system parameters can be adjusted by three factors: the number of nodes, the range of detected host, and network bandwidth.

5. Conclusions

The current service discovery system cannot deal with real-time updates and changing network services. Existing scanners only support the probing of common public protocols. This paper designed a Customizable Distributed Network Service Discovery System (CDNSDS) to solve the issue. CDNSDS consists of three subsystems: CCS, SCS, and SPS. In the CCS, a pattern set of syntax conventions is defined to assist users in customizing scan features. At the same time, the SPS provides an efficient Masscan-based scanning module. In the SPS, we describe interactive detection technology, including TCP connection management, and randomize target address in detail. Finally, the Docker Swarm Mode is used to distribute container choreography, and the experiment shows that the CDNSDS has high efficiency and accuracy, especially in industrial control protocols.

As a future work, the system should extend the syntax of the pattern set to make it better adapted to the changing protocol, such as dynamically constructing the sending packet for the reply packet. At the same time, it is necessary to calculate the relationship expressions of the transmission rate, transmission range, and the optimal ratio mathematically to arrange the distributed nodes to conduct detection with higher timeliness.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the National Key R&D Program of China (2016QY05X1000) and the National Natural Science Foundation of China (201561402137).

References

[1] W. Wu, R. Li, G. Xie et al., "A survey of intrusion detection for in-vehicle networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 919–933, 2020.

[2] L. Yin, Y. Sun, Z. Wang, Y. Guo, F. Li, and B. Fang, "Security measurement for unknown threats based on attack preferences," *Security and Communication Networks*, vol. 2018, 13 pages, 2018.

[3] A. Almohaimeed, S. Gampa, and G. Singh, "Privacy-preserving IoT devices," in *2019 IEEE Long Island systems, applications and technology conference (LISAT)*, pp. 1–5, Farmingdale, NY, USA, 2019.

[4] L. Yin, R. Li, J. Ding et al., "δ-Calculus: a new approach to quantifying location privacy☆," *Computers, Materials & Continua*, vol. 63, no. 3, pp. 1323–1342, 2020.

[5] E. Dandil, "C-NSA: a hybrid approach based on artificial immune algorithms for anomaly detection in web traffic," *IET Information Security*, vol. 14, no. 6, pp. 683–693, 2020.

[6] L. Meftah, R. Rouvoy, and I. Chrisment, *Capturing Privacy-Preserving User Contexts with IndoorHash*, A. Remke and V. Schiavoni, Eds., Distributed Applications and Interoperable Systems DAIS 2020, Springer, Cham, 2020.

[7] Z. Tian, C. Luo, J. Qiu, X. Du, and M. Guizani, "A distributed deep learning system for web attack detection on edge devices," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1963–1971, 2020.

[8] L. Yin, X. Luo, C. Zhu, L. Wang, Z. Xu, and L. Hui, "ConnSpoiler: disrupting C&C communication of IoT-based botnet through fast detection of anomalous domain queries," *Transactions on Industrial Informatics*, vol. 16, no. 2, pp. 1373–1384, 2020.

[9] M. Seliem, K. Elgazzar, and K. Khalil, "Towards privacy preserving IoT environments: a survey," *Wireless Communications and Mobile Computing*, vol. 2018, Article ID 1032761, 15 pages, 2018.

[10] X. Cao, F. Zhangjie, and X. Sun, "A privacy-preserving outsourcing data storage scheme with fragile digital watermarking-based data auditing," *Journal of Electrical and Computer Engineering*, vol. 2016, Article ID 3219042, 7 pages, 2016.

[11] S. Su, Z. Tian, S. Liang, S. Li, S. Du, and N. Guizani, "A reputation management scheme for efficient malicious vehicle identification over 5G networks," *IEEE Wireless Communications.*, vol. 27, no. 3, pp. 46–52, 2020.

[12] T. Rose, K. Kifayat, S. Abbas, and M. Asim, "A hybrid anomaly-based intrusion detection system to improve time complexity in the Internet of Energy environment," *Journal of Parallel and Distributed Computing*, vol. 145, pp. 124–139, 2020.

[13] A. V. Arzhakov and I. F. Babalova, "Analysis of current internet wide scan effectiveness," in *In young researchers in electrical and electronic engineering (EIConRus), 2017 IEEE conference of Russian*, pp. 96–99, IEEE, 2017.

[14] D. Zakir, W. Eric, and J. Alex, "ZMap: fast internet-wide scanning and its security applications," in *In Proceedings of the 22nd USENIX Security Symposium*, pp. 605–619, Washington, D.C., USA, 2013.

[15] B. A. Navamani, C. Yue, and X. Zhou, "An analysis of open ports and port pairs in EC2 instances," in *In CLOUD computing (CLOUD), 2017 IEEE 10th international conference on*, pp. 790–793, IEEE, 2017.

[16] D. Zakir, A. David, M. Ariana, and B. Michael, "R a search engine backed by internet-wide scanning," in *In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 542–553, New York, USA, 2015.

[17] R. D. Grahan, *MASSCAN: Mass IP Port Scanner*, 2013, https://github.com/robertdavidgraham/masscan.

[18] R. Rodney, J. E. Vincent, and W. P. Mark, "Large scale port scanning through tor using parallel Nmap scans to scan large portions of the IPv4 range," in *In Proceedings of the 2017 IEEE International Conferrence on Intelligence and Security Informatics*, pp. 185–187, Beijing, CHN, 2017.

[19] C. Andrei, Z. Jonas, F. Aurelien, and B. Davide, "A large-scale analysis of the security of embedded firmwares," in *In proceedings of the 23rd USENIX security*, pp. 95–110, San Diego, USA, 2014.

[20] D. Zakir, L. J. Frank, and W. Nicholas, "The matter of heartbleed," in *In Proceedings of the 14th ACM Internet Measurement Conference*, pp. 475–488, Vancouver, BC, CA, 2014.

[21] H. Nadia, D. E. Zakir, and H. Alex, "Detection of widespread weak keys in network devices," in *In Proceedings of the 21st USENIX Security Symposium*, pp. 1–21, Bellevue, WA, 2012.

[22] H. Marcella, F. Joshua, and H. Nadia, "Weak keys remain widespread in network devices," in *In Proceedings of the 14th ACM Internet Measurement Conference*, pp. 275–290, Santa Monica, USA, 2016.

[23] K. Michael and B. Joseph, "Upgrading HTTPS in mid-air: an empirical study of strict transport security and key pinning," in *In Proceedings of 2015 Network and Distributed System Security Symposium*, pp. 1–15, San Diego, USA, 2015.

[24] A. V. Arzhakov and D. S. Silnov, "Architecture of multithreaded network scanner," in *In micro/nanotechnologies and Electron devices (EDM), 2017 18th international conference of young specialists on*, pp. 43–45, IEEE, 2017.

[25] Y. Fang, X. Long, L. Liu, and C. Huang, "DarkHunter: a fingerprint recognition model for web automated scanners based on CNN," in *In proceedings of the 2nd international conference on cryptography, security and privacy*, pp. 10–15, ACM, 2018.

[26] J. Black and P. Rogaway, "Ciphers with arbitrary finite domains," in *In proceedings of the Cryptographer's track at the RSA conference 2002*, pp. 114–130, San Jose, USA, 2002.