# Identifying VPN Servers through Graph-Represented Behaviors

Chenxu Wang
Institute of Information Engineering, Chinese Academy of Sciences
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China

Jiangyi Yin*
Institute of Information Engineering, Chinese Academy of Sciences
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China

Zhao Li
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China

Hongbo Xu
Institute of Information Engineering, Chinese Academy of Sciences
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China

Zhongyi Zhang
Institute of Information Engineering, Chinese Academy of Sciences
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China

Qingyun Liu
Institute of Information Engineering, Chinese Academy of Sciences
Beijing, China

## ABSTRACT

Identifying VPN servers is a crucial task in various situations, such as geo-fraud detection, bot traffic analysis and network attack identification. Although numerous studies that focus on network traffic detection have achieved excellent performance in closed-world scenarios, particularly those methods based on deep learning, they may exhibit significant performance degradation due to changes in network environment. To mitigate this issue, a few studies have attempted to use methods based on active probing to detect VPN servers. However, these methods still have two limitations. They cannot handle situations without probing responses and are limited in applicability due to their focus on specific VPNs. In this work, we propose VPNChecker, which utilizes the graph-represented behaviors to detect VPN servers in real-world scenarios. VPNChecker outperforms existing methods in four offline datasets. The results from our datasets, containing multiple different VPNs, indicate that VPNChecker has better applicability. Furthermore, we deploy VPNChecker in an Internet Service Provider's (ISP) environment to evaluate its effectiveness. The results show that VPNChecker can improve the coverage of sophisticated detection engines and serve as a complement to existing methods.

## CCS CONCEPTS

• **Security and privacy**; • **Network security**; • **Security protocols**;

## KEYWORDS

VPN Server Detection, Active Probing, Node Classification

*Corresponding author, yinjiangyi@iie.ac.cn

**ACM Reference Format:**
Chenxu Wang, Jiangyi Yin, Zhao Li, Hongbo Xu, Zhongyi Zhang, and Qingyun Liu. 2024. Identifying VPN Servers through Graph-Represented Behaviors. In *Proceedings of the ACM Web Conference 2024 (WWW '24), May 13–17, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3589334.3645552

## 1 INTRODUCTION

Due to the growing demand for privacy protection, VPN have become increasingly popular tool [9]. Specifically, VPN can encrypt clients' network traffic and even hide clients' identities to ensure communication security. However, VPN may also be exploited to conduct geo-fraud, network attacks, malicious crawling, among other abusive situations [16, 17, 35, 38]. For example, a website may identify an incoming IP address as located in Los Angeles, while the client behind this IP may actually be located in New York. Many websites (e.g., Netflix, ChatGPT) restrict incoming IPs to protect their commercial interests. Nevertheless, VPN can help clients bypass these copyright protection strategies. Consequently, detecting VPN servers to prevent abusive activities is necessary.

In the field of VPN servers detection, much research focuses on network traffic detection [45, 50, 51]. Benefiting from advances in artificial intelligence technology, many researchers utilize machine learning and deep learning algorithms to identify anonymous servers from network traffic. In particular, those deep learning-based methods exhibit high performance in lab environments. However, Xie et al. [44] demonstrate that many deep learning-based methods might exhibit significant performance degradation when the network environment differs from the training scenarios. Their experiments indicate that some methods achieve an F1-score higher than 0.99 in the training environment, but the F1-score can fall below 0.40 in other testing situations. The sophisticated real-world network environment [5, 30, 34, 48, 49] may contain packet loss, network delay, partial network failures, and network updates. All of these network phenomena could alter traffic features, resulting in performance degradation for these traffic-based methods. The significant performance loss caused by the network environment inspires us to explore a method that does not rely on any network traffic payload information.

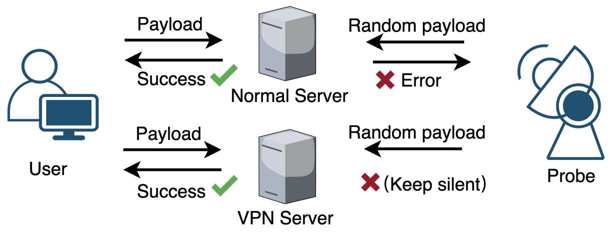

**Figure 1: Active Probing Illustration**

Recently, several works [12, 27, 47] have begun to detect VPN servers based on active probing, which is unaffected by the testing network environment. Specifically, researchers send probes to a target server and determine whether it is a VPN server based on the response information. Owing to different configuration strategies, responses from VPN servers and normal servers may be different. For instance, a normal server might return error information and close the connection after receiving an unexpected request packet, while VPN servers may remain silent until timeout, as shown in Figure 1. While previous methods based on active probing have promoted the development of VPN servers detection, they still exhibit two critical limitations: i) We discover that 16.44% of servers in our dataset do not respond with any information, and previous works cannot handle this situation. ii) They only focus on certain specific VPNs, which may lead to performance degradation when their methods are applied to other VPNs.

To overcome these limitations, we utilize graph-represented behaviors to detect VPN servers. Our design is based on the following considerations: i) Compared to VPN servers, normal servers are accessed by numerous clients and typically exhibit a more sophisticated connection relationship. We construct a communication graph to capture servers' connection behaviors. This graph relies on communication information and can detect VPN servers that lack probing response information. ii) A client might access multiple VPN servers in a short time, and these servers may share the same configuration strategy and show similar probing responses. We construct a probing graph based on this phenomenon and utilize some new features related to probing response behaviors to capture the general characteristics of VPN servers and enhance the method's detection ability.

Experimental results on four offline datasets demonstrate that our method achieves state-of-the-art (SOTA) performance. Our VPN datasets include at least 43 VPNs, and experimental results show that our method has better applicability. This paper aims to detect VPN servers in a real-world context. Therefore, we deploy our model in an ISP's environment and compare it with industrial detection engines. The results show that our model can assist these engines in identifying VPN servers.

In summary, the contributions of this work are:

**New Paradigm:** To the best of our knowledge, we are the first to combine the features of active probing and node communication relationship to detect VPN servers, offering a unique perspective on VPN server detection.

**New Technique:** We present VPNChecker, which detects VPN servers using graph-represented behaviors. VPNChecker not only exhibits better applicability but also outperforms previous methods and can enhance the coverage of sophisticated detection engines.

**New Observation:** We display some new observations regarding VPN servers, such as 'Stealth Ports'. Based on these observations, we introduce several new features, including response types and port distribution, which can assist the security community in identifying abused VPN servers.

**Ethics and Privacy:** The offline dataset in this paper is derived from the real world. To mitigate potential privacy and ethical risks, our ISP partner anonymizes all client IPs and provides us only with traffic logs, including ports, IPs, and domains. While it's true that any detection technology harbors the potential for misuse, the innovations detailed in this paper can also help the privacy community better understand network behaviors, thereby empowering efforts to safeguard user data against breaches. Any data we obtain from the detection engine is only used for evaluation purposes. Given our limited dataset, the result may not fully represent the actual capabilities of these detection engines, and our purpose is not to distinguish which is the best.

## 2 BACKGROUND

**VPN Communication Behaviors:** During the VPN communication process, a VPN client typically exhibits the following behaviors: i) Authentication: The client might need to send login information to the server for authentication, especially when utilizing commercial VPNs, such as TorGuard. ii) Fast-connection: To optimize network performance, the VPN client may attempt to connect to several servers across various locations and select the one with the shortest response time. Subsequently, the VPN client establishes a secure communication tunnel with the selected server. iii) Keep-Alive: To prevent traffic leakage [31], the VPN client may continuously send packets to ensure that the server remains active. Consequently, when communicating via the VPN tunnel, clients may connect to multiple VPN servers. These VPN servers might belong to the same vendor and share configuration strategies.

**Probing Response Behaviors:** Active probing [27] is commonly utilized to identify VPN servers. VPN servers usually exhibit one of three response behaviors after receiving probes: i) Keep Silent: If the request lacks authentication information, the VPN server remains silent and does not respond with any content [12]. ii) Expected Response: The server replies with content that matches the standard protocol. For instance, when researchers send an OpenVPN request to a server and receive a byte sequence that matches the response format [26], they can infer that this server is an OpenVPN server. iii) Unexpected Response: The server returns various error messages, non-standard protocol responses.

## 3 DESIGN OF VPNCHECKER

### 3.1 High Level Description of VPNChecker

VPNChecker is designed to identify VPN servers in real-world scenarios, which are more sophisticated than a lab environment. Given a detection time window $t$, we use $s^t_{(i,*)}$ to represent the set of servers accessed by client $i$. There are $n$ clients who access servers within a time window of $t$. VPNChecker aims to identify

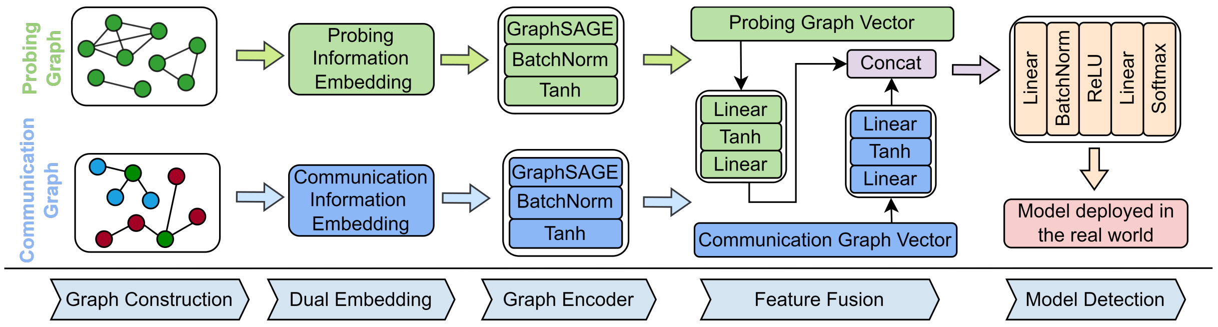

**Figure 2: VPNChecker Model Architecture**

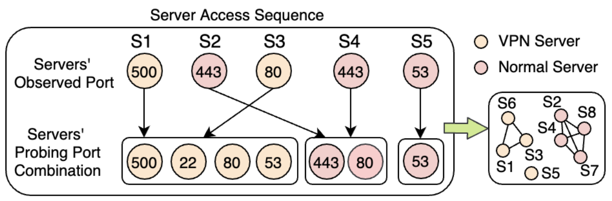

**Figure 3: Motivation of Probing Graph (S means server)**

VPN servers from this set $S$.

$$S = \sum_i^n s^t_{(i,\ vpn\ servers)} \cup \sum_i^n s^t_{(i,\ normal\ servers)}\ \ (|s^t_{(i,*)}| >= 0) \quad (1)$$

We observe that the VPN servers' probing information and communication relationship may differ from that of normal servers. For instance, normal servers might respond with error messages after receiving a TCP probe with a random payload. However, VPN servers might directly close the connection. In terms of communication relationship, normal servers are typically accessed by a myriad of clients, leading to more sophisticated connection relationships compared to VPN servers. Therefore, we utilize both the probing graph and the communication graph to capture these features and detect VPN servers.

The architecture of VPNChecker is depicted in Figure 2. We firstly construct the probing and communication graphs, embedding relevant information within their nodes. Subsequently, we encode each graph, concatenate their features, and use a classifier to identify the VPN servers. Finally, we deploy our system in the real world.

## 3.2 Probing Graph Construction

Port is widely used to reveal the services supported by servers. Given a detection time window $t$, we refer to the ports used by each server in a client's access server sequence as an *Observed Ports*, as illustrated in Figure 3. Servers might provide various services using different ports. However, the clients' server access sequences only show partial ports of a server, so collecting only the ports from the *Observed Ports* might lose some information. To this end, we conduct port scanning for each server to supplement more information. Specifically, we send TCP SYN and UDP packets to each server, which is a common method [26]. If we receive any TCP or UDP response, we determine that the port is open. We refer to the ports obtained from active probing as the *Probing Port Combination*.

Contrary to our intuition, the probing port combination of many VPN servers is not random. We assume that if a port combination appears fewer than 10 times, it may be random. Such combinations account for only 18.47% of the total. Furthermore, we discover that servers belonging to the same vendor may have similar probing port combinations. This might be because servers from the same provider often share a deployment strategy. For example, Psiphon3 servers typically use the probing port combination {443, 53, 22}, {443, 554, 22}, and so on. For more details, please refer to Appendix A. This phenomenon inspires us to construct a graph using this similarity.

Given the time window $t$, we assume that the server $s_j$ and $s_k$ in the set $s^t_{(i,*)}$ use similar probing port combination $\mathcal{P}(s_j)$ and $\mathcal{P}(s_k)$, indicating they offer similar services. This similarity $\zeta$ is computed by using formula $\mathcal{J}$. We use $\mathcal{PG}_t$ to describe this relationship, where $\mathcal{E}^t$ and $\mathcal{D}^t$ represent the edge and node sets, respectively, and $\mathcal{N}(m)$ signifies the set of nodes that connect to node $m$. Please note that not every server has a connected node. To facilitate subsequent computations, we add self-loops for such isolated nodes. The node features of this graph are mainly derived from the active probing, therefore we refer to $\mathcal{PG}_t$ as the probing graph.

$$\mathcal{J}(\mathcal{P}(s_j)\ ,\ \mathcal{P}(s_k)) = \frac{|\mathcal{P}(s_j)| \cap |\mathcal{P}(s_k)|}{|\mathcal{P}(s_j)| \cup |\mathcal{P}(s_k)|} \quad (2)$$

$$\mathcal{E}^t_i = \left\{(s_j,\ s_k)|\ s_j,\ s_k \in s^t_{(i,*)}\ and\ \mathcal{J}(\mathcal{P}(s_j)\ ,\ \mathcal{P}(s_k)) >= \zeta\right\} \quad (3)$$

$$\mathcal{E}^t = \cup^n_{i=1} \mathcal{E}^t_i \qquad \mathcal{D}^t = \cup^n_{i=1} s^t_{(i,*)} \quad (4)$$

$$\mathcal{E}_{self-loop} = \left\{(m,\ m) \mid m \in \mathcal{D}^t\ and\ m \notin \mathcal{N}(m)\right\} \quad (5)$$

$$\mathcal{PG}_t = <\mathcal{D}^t,\ \mathcal{E}^t \cup \mathcal{E}_{self-loop}> \quad (6)$$

Our probes include application layer probes and transport layer probes. From the application layer probes, we extract features related to response types. Based on the transport layer probes, we derive features concerning response time, response length, termination state, and port distribution.

**Application Layer Probes:** Application layer probes include both traditional VPN protocol probes and popular protocol probes. Our design is based on the following observation: i) Many VPN servers still utilize traditional protocols [27], such as PPTP, IPSec, SSTP, and OpenVPN. In our dataset, at least 27.15% of VPN servers deploy these traditional VPN protocols. Consequently, we believe that traditional VPN protocols probes can help us discover VPN servers. ii) To bypass detection, many VPN servers disguise their traffic to popular protocols and use the standard ports [32, 36, 40].

Specifically, our traditional VPN protocol probes and popular protocol probes encompass SSTP, IPSec, OpenVPN, PPTP, FTP, SSH, DNS, HTTP and TLS. We employ these probes to obtain response types for each server. To accelerate probing speed, all of application layer probes are sent to standard ports only. An overview of probe content is shown in Table 1. The response types reveal not only whether the server supports the corresponding protocol but also the server's deployment strategy. Below are two examples of our probes and we refer interested readers to [3] for more details.

**OpenVPN (TCP) Response Type:** We send a TCP packet containing `\x38{8}\x00\x00\x00\x00` to a target server. If the server responds with `\x00{1}\x40{*}\x00\x00\x00\x00`, it strongly suggests that the server is running an OpenVPN service [27] since this response content matches the standard OpenVPN format. We set this response type as $\mathcal{RT}_o = 1$ ($o$ means OpenVPN). Previous works [12, 47] have shown that VPN servers may employ configurations to bypass probing attempts. We observe that VPN servers might have the following response types after they receiving an OpenVPN probe: i) The server directly rejects the TCP connection, indicated by $\mathcal{RT}_o = 2$. ii) The server remains silent until a timeout occurs, which is represented as $\mathcal{RT}_o = 3$. iii) The server responds with an empty packet after establishing a TCP connection, denoted by $\mathcal{RT}_o = 4$. We observe that while many servers utilize port 1194 for communication in the real world, only 3.76% of those return a standard OpenVPN protocol response. This suggests that many VPN servers adopt probe-resistant strategies. Therefore, a detailed analysis of response type is necessary. All of these response types enable us to understand the server in greater detail than simply focusing on whether a response is received.

Although popular probing tools like Nmap are also capable of probing VPN protocols, existing research [21, 24] suggest that servers might filter their probes due to exposed probing fingerprints. Therefore, we design probes using a public Python library [11] instead of using existing probing tools. Our probing result indicates that our VPN protocol probes have a response success rate about twice that of Nmap.

**DNS Response Type:** The DNS protocol allows clients to assign a DNS resolver to query the IP addresses of domains. We consider that the response behaviors of VPN servers, which mimic DNS traffic and use port 53, may differ from normal servers. To distinguish between VPN and normal servers, we assign the target server as a DNS resolver and send a DNS query about `google.com` (a common test domain). The response types of servers include: i) The server answers with the IPs of `google.com`, indicating it is a DNS server, denoted as $\mathcal{RT}_d = 1$ ($d$ means DNS). ii) The server responds with `All nameservers failed to answer`, represented as $\mathcal{RT}_d = 2$. Our intuition is that `google.com`, being one of the most globally popular domains, is likely cached in a DNS resolver. We discover that 34.24% of responsive VPN servers fail to answer, while only 18.89% of responsive normal servers provide that response. This difference can help us detect VPN servers. iii) The server does not reply with any response until a timeout occurs, referred to as $\mathcal{RT}_d = 3$. All of these response types can reveal a server's configuration for port 53.

In general, we utilize these application layer probes to gather response type features for each target server. These features can disclose server configurations and aid us in detecting VPN servers.

**Table 1: Summary of Application Layer Probes**

| Name | UDP/TCP | Probe Content | Port |
|---|---|---|---|
| SSTP | TCP | `\x53\x53\x54\x50\x5F...` | 443 |
| IPSec | UDP | `{8}\x01\x10...` | 500 |
| OpenVPN | TCP&UDP | `\x38{8}\x00\x00\x00\x00` | 1194 |
| PPTP | TCP | `\x00\x9c\{6}\x00\x01...` | 1723 |
| FTP | TCP | `Python TCP Connection` | 21 |
| SSH | TCP | `Python TCP Connection` | 22 |
| DNS | UDP | `Query google.com` | 53 |
| HTTP | TCP | `GET / HTTP/1.1\r\n...` | 80 |
| TLS | TCP | `Python SSL Connection` | 443 |

**Transport Layer Probes:** The transport layer probes encompass both TCP and UDP probes. We extract the features of response time, response length, termination state and port distribution based on these probes.

**Response Time and Length:** As defined earlier, refer to the ports observed in the ISP's environment as *Observed Ports*. We probe each *Observed Port* using TCP and UDP probes with 1500 bytes (standard Maximum Transmission Unit (MTU)) to collect the total packet length ($\mathcal{PL}$), packet count ($\mathcal{PC}$) and time duration ($\mathcal{DT}$, calculated as the difference between the last and first response times). To accelerate the probing process, we set the maximum timeout period to 300 seconds. For servers with multiple *Observed Ports*, we select results that exhibit the maximum values for $\mathcal{PL}$, $\mathcal{PC}$ and $\mathcal{DT}$. We observe that these features may differ between VPN and normal servers. For instance, VPN servers prefer to reply with a longer response time and a shorter response packet after receiving a TCP probe, as shown in Figure 4 and Figure 5. We believe this may be a strategy for VPN to reduce the exposure risk.

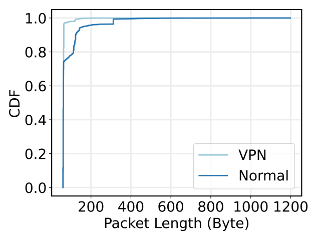

**Figure 4: Response Length**

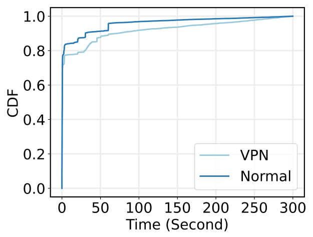

**Figure 5: Response Time**

**Termination State:** TCP servers might terminate a connection if they receive bytes that cannot be parsed. Previous research [12] has indicated that receiving bytes beyond a certain threshold can lead a server to send a FIN packet, while a Linux server that closes a TCP connection with unread bytes in its buffer might trigger an RST packet. The RST/FIN threshold represents the minimum byte count that triggers an RST/FIN packet. Existing work [47] has shown that the RST threshold distribution differs between VPN and normal servers. Therefore, while some servers might close the connection and not respond with content to avoid discovery, their termination behavior can also be utilized to detect VPN servers.

We consider that the RST/FIN threshold is influenced by a server's configuration strategy. Drawing from prior works [7, 12, 47], we utilize the RST and FIN thresholds as features to detect VPN servers. We randomly send TCP probes to estimate these thresholds for each server. We assume the value of RST/FIN threshold is smaller than the standard MTU. Notably, while the MTU is 1500 bytes, sending 1500 probes is unnecessary. We can use the binary search method to optimize probing. The scope of our probe ports contains *Observed Ports* and the ports enumerated in Table 1. To ensure consistent feature dimensions and facilitate subsequent calculations, for servers with multiple *Observed Ports* entries, we select one with the smallest RST/FIN threshold. If this port matches the ports in Table 1, we also record its value.

**Port Distribution:** We perform port scanning for every server and refer to the ports obtained by probing as *Probing Ports*. We discover that servers may use multiple ports to communicate with clients in the ISP's environment. However, some of these ports do not respond to any message after receiving our probe, not even with a message to close the connection. We compute the difference in ports ($\mathcal{DP}$, $\mathcal{DP} = Observed\ Ports - Probing\ Ports$) to describe this phenomenon, which we refer to as 'Stealth Ports'. In other words, $\mathcal{DP}$ means the ports provide service for clients but prohibits probing. In our dataset, the average length $\mathcal{DP}$ of VPN servers is about 2.5 times that of normal servers. This difference may be due to some VPN servers adopting a more strict probe-resistant strategy. Additionally, we also compute the number of $|Observed\ Ports|$ and $|Observed\ Ports \cup Probing\ Ports|$ as features to detect VPN servers.

The port distribution sequence of server may reveal the services supported by the server. Our dataset indicates that approximately 90% of servers utilize fewer than 35 ports. Therefore, such sparse distribution (35/65535) poses a challenge to feature representation. We design the following formula to compress the port distribution into a 10-dimensional feature vector $\mathcal{V}$. The intention of this formula is to retain some information about the server's port patterns while compressing the features. Where $I$ is an indicator function, $p_j$ is port $j$ from *Observed Ports* $\cup$ *Probing Ports*. The indicator function takes the value 1 when two ports are the same and 0 when they are different.

$$\mathcal{V} = \mathcal{V}_0 \oplus \mathcal{V}_1 \oplus ... \oplus \mathcal{V}_9 \tag{7}$$

$$\mathcal{V}_i = \sum_{j=1}^{m} \mathcal{I}(p_j \ mod\ 10, i) \tag{8}$$

Generally, we obtain features derived from response types, response time, response length, termination state, and port distribution for every server using the aforementioned method. As defined earlier, the probing graph is referred to as $\mathcal{PG}_t$. Given the $\mathcal{PG}_t$, we utilize the aforementioned features as initial node features $\mathcal{F}_r$. Subsequently, we use GraphSAGE $\mathcal{M}$ to refine these node features through neighbor feature aggregation and use the *lstm* as the aggregation function. The final features $\mathcal{A}_r(u)$ for each node $u$ in this graph are as follows:

$$\mathcal{A}_r(u) = \mathcal{M}_{lstm}(\mathcal{F}_r(u), \mathcal{H}_r(u)) \tag{9}$$

Where $\mathcal{F}_r(u)$ represents the initial features of $u$ and $\mathcal{H}_r(u)$ represents the neighbors' features of node $u$. As the neighboring nodes in this graph provide similar services, we consider that the aggregation operation can enhance the feature representation.

## 3.3 Communication Graph Construction

We gather client IPs, server IPs, and ports from the ISP's environment. Additionally, we collect the domains of each server from its Pointer Record (PTR) and the certificate's Common Name (CN). These domains have been frequently utilized in server analysis. For each server, we query their certificate from port 443 and *Observed Ports*. We note that despite the certificate being defaultly deployed on port 443, about 18.40% of certificates are deployed on other ports. Therefore, it is necessary to consider the *Observed Ports*. Given an observation time window $t$, we construct a graph $\mathcal{CG}_t$ to describe the communication relationships among servers. Where $\mathcal{L}_i$ represents a node of client IP, server IP, and domain, $h$ denotes the number of nodes, and $\mathcal{R}(\mathcal{L}_i)$ signifies the edges between node $\mathcal{L}_i$ and others. we refer to this graph as the communication graph.

$$\mathcal{CG}_t = < \bigcup_{i=1}^{h} \mathcal{L}_i,\ \bigcup_{i=1}^{h} \mathcal{R}(\mathcal{L}_i) > \tag{10}$$

We consider that the communication behaviors of VPN servers may be different from that of normal servers. Additionally, previous work [4] has show that topological features are beneficial in graph classification. Therefore we use topological features as the initial node embedding vector in $\mathcal{CG}_t$. The topological features are derived from degree, eigenvector centrality, pagerank, closeness centrality, local clustering coefficient and K-core. These metrics can reveal the intrinsic structural properties and significance of nodes within a network, providing a representation of the node relationships. These metrics are widely used in graph analysis [29] and the code for constructing these features can be found in public Python libraries [14], more details can be found in Appendix B.

We use the GraphSAGE $\mathcal{M}$ to capture the features of each node and choose the aggregation function to be *mean*. In this graph, we use $\mathcal{A}_c(v)$ to describe the feature of node $v$. Where $\mathcal{F}_c(v)$ represents the initial features of node $v$ and $\mathcal{H}_c(v)$ represents the neighbors' features of node $v$.

$$\mathcal{A}_c(v) = \mathcal{M}_{mean}(\mathcal{F}_c(v), \mathcal{H}_c(v)) \tag{11}$$

## 3.4 Detection Model

We use the probing graph and the communication graph to represent probing response behaviors and communication behaviors, respectively. For target server $x$, we obtain the features from the probing graph and the communication graph using the methods described above. Subsequently, we use the formula $\mathcal{Y}(x)$ to enhance their features and then concatenate all the features together. Where $\mathcal{R}(x)$ represents a linear layer, containing learnable parameters $\mathcal{W}$ and $b$, $\mathcal{R}(x)_*$ denotes a linear layer with its own set of learnable parameters $\mathcal{W}$ and $b$. Finally, we use $\mathcal{Z}(x)$ to predict servers.

$$\mathcal{R}(x) = \mathcal{W}x + b \tag{12}$$

$$\mathcal{Y}(x) = \mathcal{R}_1 tanh \mathcal{R}_2(\mathcal{A}_c(x))\ \ CONCAT\ \ \mathcal{R}_3 tanh \mathcal{R}_4(\mathcal{A}_r(x)) \tag{13}$$

$$\mathcal{Z}(x) = Softmax(\mathcal{R}_5 ReLu(\mathcal{R}_6(\mathcal{Y}(x))) \tag{14}$$

## 4 EXPERIMENTS

### 4.1 Dataset

**Offline Dataset:** Maghsoudlou et al. [27] disclose that VPN traffic accounts for at least 2.6% in their ISP's environment. There are million of servers in our ISP's environment within one day. Consequently, collecting and labeling VPN servers becomes a significant challenge. Fortunately, previous work [31] has indicated that many VPN servers are centrally deployed in some Autonomous Systems (AS), such as 9009 and 60068. We obtain these servers'[31] labels from our industry partner, who claims that their labels are derived from threat intelligence and their detection model. Our partner can identify various server types including VPN servers, Proxy servers, and normal servers. To expedite the data collection process, we regard VPN servers in the aforementioned ASs as seed IPs.

We present a new context-based dataset collection method that assists researchers in collecting VPN servers in the real world. The data collection procedure is as follows: i) Sequence collection: Given a time window $t$, we record the timestamp $\mathcal{T}_i$ when client $i$ accesses the seed IPs. Owing to IP-sharing access technologies, such as Network Address Translation (NAT) [33], multiple endpoints might be behind one client IP. VPN software allows a portion of application traffic to be routed through the VPN tunnel, while it routes other traffic outside this tunnel [52]. As a result, clients may access multiple VPN servers within a short time. For client $i$, we gather the server access sequence during the $2\varphi$ seconds $[\mathcal{T}_i - \varphi, \mathcal{T}_i + \varphi]$. ii) Server collection: Since the VPN servers in the same server access sequence may be deployed on different ASs, we can collect many VPN servers from this window. iii) Data Processing: The client may access numerous server IPs within a short time. For instance, we observe that some IPs even access over 1,000 server IPs in just 10 seconds. To eliminate these abnormal sequences, we choose only the server access sequences with a length smaller than $\varrho$.

**Table 2: Offline Dataset Distribution**

| Category | Count | Percent |
|---|---|---|
| VPN | 6840 | Psiphon3 6.64%, NordVPN 1.09%, PIA 0.92%, ExpressVPN 0.05%, Surfshark 0.04%, Others 5.20% |
| Proxy | 6295 | 13.54% |
| Normal | 33346 | 71.74% |
| Total | 46481 | 100% |

The distribution of the offline dataset is shown in Table 2. We organize these data into four datasets: VPN and normal servers ($\mathcal{D}_1$), popular VPN and normal servers ($\mathcal{D}_2$), unpopular VPN and normal servers ($\mathcal{D}_3$), and Proxy and normal servers ($\mathcal{D}_4$).

There are more than 43 (refer to Table 7 in Appendix) different VPN servers in our dataset. The servers of the top three VPNs (Psiphon3, NordVPN, PIA) account for 55.95% of all VPN servers, and these VPNs have a certain probe-resistant ability.

We use $\mathcal{D}_2$ to enable the model to focus on features of popular VPNs and to demonstrate the model's identification ability. We employ $\mathcal{D}_3$ to learn features across a wide range of VPNs, showcasing its broad applicability. Considering the partial functional similarity between Proxies with VPNs, we also use $\mathcal{D}_4$ to explore the model's potential detection ability to Proxies. We refer interested readers to [3] for more details about the datasets.

**Online Dataset:** To test the performance of our model in the online environment, we deploy it in the environment of our ISP partner. Unlike the offline dataset, we do not specify seed IPs during the data collection process. We collect 65,636 client access sequences, which contain 512,170 server IPs.

### 4.2 Offline Experiments

**Implementation Details:** During the data collection stage, we set an observation time window $\varphi$ to 10 seconds, the largest server sequence access length $\varrho$ to 50 and the *Probing Port Combination* similarity $\zeta$ to 0.5 by default. In the subsequent model training stage, the maximum training epoch is set to 150. We choose Adam as the optimizer and set the learning rate to 0.005. Each experiment is conducted 10 times, with the results being averaged for display. To eliminate the impact of data imbalance, we use the downsampling strategy (randomly discarding samples) to ensure consistency between the positive and negative sample. The models are evaluated by utilizing Accuracy (AC), Precision (PR), Recall (RC), and F1-score (F1). As we have mentioned, servers may not respond with any information after receiving probes. In this situation, we set the corresponding feature value of each model to 0. In our offline dataset, 16.44% of servers do not provide any probing response information.

**Comparison Experiments:** Although much work has been done in the field of VPN server detection, it primarily focuses on traffic information. These methods extract features based on traffic payload, which may be affected by the network environment. In our scenario, we primarily extract features from active probing. Our goal is not to replace traffic-based methods, but to demonstrate that not using traffic features can also be feasible. To this end, we only select those methods that are based on active probing for comparison.

The results are shown in Table 3. OOVF [47] focuses on the servers that use the OpenVPN protocol and primarily uses the features related to response time to detect VPN servers. DPP [12] aims to identify servers deploying probe-resistant protocols, principally extracting features from RST/FIN thresholds, response time, and response content. The aforementioned studies have greatly inspired our work. However, since their methods primarily focus on specific VPN(s), and result in decreased performance on $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$. Since OOVF only focuses on the OpenVPN protocol, it has the worst performance on $\mathcal{D}_4$. ACER [7] is designed to detect shadowsocks proxy, so it performs better than OOVF and DPP in $\mathcal{D}_4$. ACER extracts features from response time, TCP flags, and packet information created during active probing. This method is also adaptable for detecting VPN servers. Our model can also use GCN [20], GAT [37], GIN [46], SGC [42] to aggregate graph features. We also show their performance in Table 3. The results show that VPNChecker achieves the highest F1-score on all datasets.

**Ablation Experiments:** As our detection target is VPN servers, we conduct ablation experiments on $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$. The results are displayed in Table 4. To facilitate presentation, we denote the probing graph as *PG*, the communication graph as *CG*, and *W* signifies 'without'. The results clearly indicate that both *PG* and *CG* are beneficial for detection, with *PG* contributing most significantly. This demonstrates that probing features are more important than topological features. The results of $W/G$ reveal that our model surpasses previous methods by solely utilizing features from response

**Table 3: Offline Experiments**

| Model | $\mathcal{D}_1$: ALL VPN and Normal | | | | $\mathcal{D}_2$: Popular VPN and Normal | | | | $\mathcal{D}_3$: Unpopular VPN and Normal | | | | $\mathcal{D}_4$: Proxy and Normal | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | AC | PR | RC | F1 | AC | PR | RC | F1 | AC | PR | RC | F1 | AC | PR | RC | F1 |
| OOVF [47] | 0.7793 | 0.8350 | 0.7042 | 0.7641 | 0.8282 | 0.8555 | 0.8018 | 0.8278 | 0.7049 | 0.7856 | 0.5718 | 0.6619 | 0.5369 | 0.6040 | 0.2425 | 0.3460 |
| DPP [12] | 0.6559 | 0.6224 | 0.8018 | 0.7008 | 0.7045 | 0.6517 | 0.8927 | 0.7534 | 0.6284 | 0.6125 | 0.7963 | 0.6924 | 0.7047 | 0.6933 | 0.7903 | 0.7386 |
| ACER [7] | 0.6788 | 0.7042 | 0.6337 | 0.6337 | 0.7052 | 0.6692 | 0.6692 | 0.7530 | 0.7269 | 0.7468 | 0.7074 | 0.7266 | 0.8176 | **0.8118** | 0.8450 | 0.8281 |
| GCN | 0.8353 | 0.8878 | 0.7675 | 0.8233 | 0.8575 | 0.8549 | 0.8612 | 0.8581 | 0.8374 | 0.9119 | 0.7467 | 0.8211 | 0.8071 | 0.7553 | 0.9084 | 0.8248 |
| GAT | 0.8307 | 0.8788 | 0.7671 | 0.8192 | 0.8673 | **0.9616** | 0.7650 | 0.8521 | 0.8358 | 0.8938 | 0.7623 | 0.8229 | 0.8064 | 0.7611 | 0.8932 | 0.8218 |
| GIN | 0.8404 | 0.8337 | 0.8504 | 0.8420 | 0.8451 | 0.7928 | **0.9343** | 0.8577 | 0.8410 | **0.9474** | 0.7222 | 0.8196 | 0.8302 | 0.7735 | 0.9337 | 0.8461 |
| SGC | 0.8333 | **0.9009** | 0.7490 | 0.8180 | 0.8710 | 0.9536 | 0.7798 | 0.8580 | 0.8399 | 0.9156 | 0.7487 | 0.8238 | 0.8180 | 0.7728 | 0.9006 | 0.8318 |
| VPNChecker | **0.8565** | 0.8382 | **0.8835** | **0.8603** | **0.8932** | 0.8864 | 0.9019 | **0.8941** | **0.8672** | 0.8949 | **0.8323** | **0.8625** | **0.8447** | 0.7915 | **0.9356** | **0.8575** |

**Table 4: Ablation Experiments**

| Method | $\mathcal{D}_1$: ALL VPN and Normal | | | | $\mathcal{D}_2$: Popular VPN and Normal | | | | $\mathcal{D}_3$: Unpopular VPN and Normal | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Accuracy** | **Precision** | **Recall** | **F1** | **Accuracy** | **Precision** | **Recall** | **F1** | **Accuracy** | **Precision** | **Recall** | **F1** |
| W/G | 0.8399 | 0.8807 | 0.7903 | 0.8330 | 0.8890 | 0.9248 | 0.8508 | 0.8863 | 0.8399 | 0.8803 | 0.7870 | 0.8311 |
| W/PG | 0.5582 | 0.5334 | **0.9288** | 0.6777 | 0.5675 | 0.5411 | 0.8890 | 0.6727 | 0.6279 | 0.5866 | **0.8673** | 0.6999 |
| W/CG | 0.8511 | 0.8492 | **0.8538** | 0.8515 | 0.8529 | 0.9739 | 0.7253 | 0.8314 | 0.8363 | 0.8616 | 0.8616 | 0.8616 |
| W/lstm | 0.8397 | **0.9100** | 0.7539 | 0.8246 | 0.8719 | 0.9774 | 0.7613 | 0.8560 | 0.8235 | 0.8481 | 0.7878 | 0.8169 |
| W/tanh | 0.8531 | 0.8744 | 0.8246 | 0.8488 | 0.8691 | 0.9807 | 0.7530 | 0.8519 | 0.8286 | 0.9209 | 0.7191 | 0.8076 |
| W/MLP | 0.8579 | 0.8685 | 0.8436 | 0.8559 | 0.8626 | 0.9816 | 0.7391 | 0.8433 | 0.8291 | 0.8800 | 0.7623 | 0.8170 |
| Linear | 0.8584 | 0.8636 | 0.8514 | 0.8574 | 0.8641 | 0.8872 | 0.8342 | 0.8599 | 0.8384 | 0.8680 | 0.7984 | 0.8317 |
| CNN | **0.8621** | 0.8973 | 0.8177 | 0.8557 | 0.8774 | 0.9105 | 0.8372 | 0.8723 | 0.8590 | 0.8930 | 0.8158 | 0.8527 |
| LSTM | 0.8246 | 0.8850 | 0.7461 | 0.8096 | 0.8645 | **0.9828** | 0.7419 | 0.8455 | 0.8224 | 0.8903 | 0.7353 | 0.8054 |
| VPNChecker | 0.8565 | 0.8382 | 0.8835 | **0.8603** | **0.8932** | 0.8864 | **0.9019** | **0.8941** | **0.8672** | **0.8949** | 0.8323 | **0.8625** |

types, response time, response length, termination state, and port distribution. This indicates that our extracted features are more efficient. In the $PG$, we use $lstm$ as the aggregation function in GraphSAGE. The $W/lstm$ means we use the default method $mean$. The better result of $lstm$ might be attributed to the $lstm$ aggregation function's superior information mining ability compared to $mean$. We also conduct ablation experiments about replacing $tanh$ with $relu$ and removing $MLP$ in feature fusion. The results show that all the structures are necessary. Additionally, the results from using a Linear layer, $CNN$, $LSTM$ as classifiers indicate that our classifier is more suitable.

**Sensitivity Experiments:** We conduct sensitivity experiments on sequence length $\varrho$, observation time $\varphi$ and similarity threshold $\zeta$ as mentioned in Implementation Details section. Figure 6-a illustrates that shorter client access sequence lengths correlate with higher F1-score. This may be because shorter sequences contain a higher proportion of VPN servers. In the construction process of the $PG$, we observe that VPN servers might utilize the same port combinations as normal servers, which can lead to misclassifications. When the proportion of VPN servers increases, such misclassifications are reduced. Figure 6-b shows that the model is not sensitive to time $\varphi$. This may be because the data distribution is consistent across different time windows. Figure 6-c indicates that changes in the similarity threshold do not significantly impact the results, possibly because some VPN have a stable port combination.

## 4.3 Online Experiments

We deploy our system in an ISP's environment and analyze 512,170 servers, identifying 6,143 suspicious VPN servers among these. We compared it with four industry detection engines: IPQualityScore [17], VPNAPI.io [38], SPUR [35], IPInfo [16] as detailed in Table 5.

**Ground truth construction:** i) We collect domain names associated with servers. Specifically, we query the PTR records and

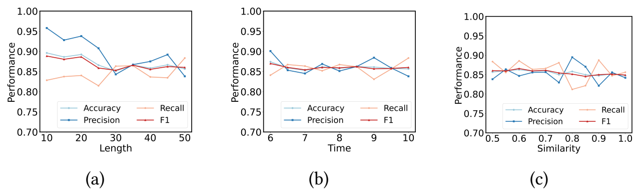

**Figure 6: (a), (b), (c) show the result of Sequence Length, Sequence Time and Similarity, respectively.**

**Table 5: Online Experiments**

| **Industry Engine** | **Count** | **Coverage Ratio** |
|---|---|---|
| IPQualityScore [17] | 4403 | 0.9724 |
| VPNAPI.io [38] | 4315 | 0.8937 |
| IPInfo [16] | 766 | 0.1587 |
| SPUR [35] | 3074 | 0.6367 |
| Ground truth | 4528 | 1.0000 |

certificate SANs using Linux commands and gather HTTP host and TLS SNI records from the ISP environment. ii) Following the method outlined in [9], we identify VPN servers by checking for the 'vpn' character in their domain names (e.g., vpngate.net). In this step, we obtain 211 VPN servers. iii) We collect popular VPNs' domains, such as surfshark.com and torguard.org. If a server deploys a VPN domain, we consider it a VPN server. From this step, we acquire 146 VPN servers. iv) Using the method from [27], we identify VPN servers by sending VPN protocol probes, resulting in 453 VPN servers. v) IPQualityScore, VPNAPI.io, and SPUR are designed to identify VPN, proxy, and Tor servers, so they have more powerful identification capabilities. IPInfo offers multi-dimensional IP data, but its capacity to detect VPN servers is limited to only a few instances. We adopte a simple voting strategy for ground truth selection: we only select servers that are marked as VPN by at least two of IPQualityScore, VPNAPI.io, and SPUR. In this step, we obtain 4396 VPN servers. Totally, we obtain 4528 deduplicated VPN servers from step ii) to step v).

**False alarms analysis:** VPNChecker identifies 6143 VPN servers. We check the 1615 (6143-4528) IPs and discover 522 IPs identified as proxy or Tor by detection engines. This because proxy/Tor servers often have similar features to VPN servers. Considering that these servers may also be abused, we think that these are not false alarms. For the remaining 1093 (1615-522) IPs, we try our best to analyze them, and there is no evidence to indicate that they are VPN servers. Specifically, we check whether an IP is in the VPN/abuse IP range provided by [1, 2, 19, 31]. For the remaining 1093 (1615-522) IPs, we count the number of visits to these servers within one hour. Among them, 205 Server IPs have been visited less than 10 times. We believe that the risk of these IPs being false alarms is very low.

Among the 6143 VPN servers identified by VPNChecker, 4528 are VPN, 522 are proxy/Tor servers and 1093 IPs are vague IPs. Our online experiments show that VPNChecker can identify more VPN IPs detection engines. Therefore, our method can help industrial engines improve their detection capabilities.

## 5 DISCUSSION AND LIMITATIONS

**Adaptive attacks:** VPNChecker outperforms other models across various datasets, partially due to the possibility that attackers have altered their tactics in response to previous detection methods. Detection and anti-detection form a continual game of chase. Attackers may refine their strategies to reveal fewer server features after our method becomes public. For instance, they might randomize the RST/FIN threshold. However, this requires modifying the operating system's source code. Attackers can also alter communication behaviors to disrupt the node connection relationships within our graph, but this could compromise user experience. In general, while attackers can employ strategies to bypass our detection, these strategies are likely to incur additional costs.

**Limitation:** In this paper, we mainly focus on the features of active probing and topology. We assume that security researchers have the ability to receive probe responses and obtain node connection relationships. However, our method may be limited in some scenarios. If researchers conduct server detection at the gateway, they may not be able to obtain complex node interaction relationships. In this situation, although our method can still be applied, ablation experiments indicate that our model's performance would decline. If researchers conduct detection in situations where internet access is not available, our method is not applicable.

## 6 RELATED WORK

### 6.1 Based on Network Traffic

Numerous works focus on network traffic detection and almost of them can be used to identify VPN servers. Several studies [6, 8, 28, 41, 43] extract features from various statistical information, such as time intervals, packet lengths, and byte entropy, and then employ machine learning or rule-based methods for detection. For instance, Wu et al. [43] suggest that while servers may randomize traffic to evade detection, researchers can use byte entropy to identify traffic. Some research [23, 45, 50, 51] centers on feature extraction from packet length sequences and arrival interval sequences, utilizing deep learning methods for classification. For example, Jiang et al. [18] use sequence information to construct traffic graphs, subsequently employing Graph Neural Networks for identification. Some studies [22, 25] utilize raw packets as features for detection, achieving excellent classification results. Although the above works achieve excellent classification results in the training environment, these methods may have performance loss due to the changes in environment. For example, Xie et al. [44] show that some methods might show performance degradation in different networks.

### 6.2 Based on Active Probing

**Traditional Protocols:** Traditional VPN protocols, such as IPsec [15] and PPTP [13], are still widely used[19, 27]. These protocols are open-source and primarily designed to keep communications secure rather than to resist detection. Owing to their open-source and non-resistant nature, researchers can easily construct VPN requests and send probe packets to target servers. By analyzing the response content, they can determine whether the server provides VPN services. Maghsoudlou et al. [27] use this method to discover more than 9.8M VPN servers on the Internet.

**Probe-Resistant Protocols:** Some studies [12, 47] have pointed out that many VPN servers deploy probe-resident protocols, such as Psiphon's obfuscation SSH protocol. These servers reply only to requests that include authentication information. Since it is difficult to acquire authentication information, researchers cannot receive any response after sending probing packets. Frolov et al. [12] observe that probing with popular protocols, such as HTTP, can effectively filter most normal servers out. Additionally, they find that TCP timeout and data thresholds can be used to distinguish VPN servers from others. Xue et al. [47] discover that some commercial OpenVPN servers also deploy authentication mechanisms and they utilize the timeout and RST thresholds to detect these OpenVPN servers. Some studies [7, 10, 39] are not directly aimed at detecting VPN servers but offer methods that could be applied to detect VPN servers. Fifield et al. [10, 39] indicate that HTTP and TLS response may aid in classification. Cheng et al. [7] suggest that researchers could focus on metrics such as timeout, TCP flags and response packet length.

## 7 CONCLUSION

In this paper, we introduce a new method for detecting VPN servers using a probing graph and a communication graph. We present some interesting features, such as response types and port distribution, which can help the security community enhance server detection capabilities. Our experimental results demonstrate that using features of active probing and topological relationships is feasible. The results also show that our method outperforms previous methods and can help industrial detection engines enhance coverage ratios.

## ACKNOWLEDGMENTS

This work is supported by the Scaling Program of Institute of Information Engineering, CAS (Grant No. E3Z0041101 and No. E3Z0191101).

## REFERENCES

[1] 2023. FireHOL. iplists.firehol.org. Accessed: 2023-10-05.
[2] 2023. IBlockList. www.iblocklist.com/lists. Accessed: 2023-10-05.
[3] 2023. VPNChecker. https://github.com/chenxuStep/VPNChecker. GitHub repository.
[4] Roy Abel, Idan Benami, and Yoram Louzoun. 2019. Topological based classification using graph convolutional networks. *CoRR* abs/1911.06892 (2019).
[5] Mohammed Alfatafta, Basil Alkhatib, Ahmed Alquraan, and Samer Al-Kiswany. 2020. Toward a generic fault tolerance technique for partial network partitioning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 351–368.
[6] Julian Andres Caicedo-Munoz, Agapito Ledezma Espino, Juan Carlos Corrales, and Alvaro Rendon. 2018. QoS-Classifier for VPN and Non-VPN traffic based on time-related features. *Computer Networks* 144 (2018), 271–279.
[7] Jiaxing Cheng, Ying Li, Cheng Huang, Ailing Yu, and Tao Zhang. 2020. ACER: detecting Shadowsocks server based on active probe technology. *Journal of Computer Virology and Hacking Techniques* 16 (2020), 217–227.
[8] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A Ghorbani. 2016. Characterization of encrypted and vpn traffic using time-related. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*. 407–414.
[9] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Enric Pujol, Ingmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, et al. 2020. The lockdown effect: Implications of the COVID-19 pandemic on internet traffic. In *Proceedings of the ACM internet measurement conference*. 1–18.
[10] David Fifield. 2017. *Threat modeling and circumvention of Internet censorship*. University of California, Berkeley.
[11] Python Software Foundation. 2023. *socket — Low-level networking interface*. https://docs.python.org/3/library/socket.html (Accessed on 2023-09-04).
[12] Sergey Frolov, Jack Wampler, and Eric Wustrow. 2020. Detecting Probe-resistant Proxies.. In *NDSS*.
[13] Network Working Group. 1993. *The Point-to-Point Protocol (PPP)*. https://www.rfc-editor.org/rfc/rfc2637.html (Accessed on 2023-09-03).
[14] Aric A. Hagberg, Daniel A. Schult, Pieter J. Swart, et al. 2023. *NetworkX*. https://networkx.org/ (Accessed on 2023-09-04).
[15] Internet Engineering Task Force (IETF). 2011. *IP Security (IPsec) and Internet Key Exchange (IKE)*. https://www.rfc-editor.org/rfc/rfc2637.html (Accessed on 2023-09-03).
[16] Ipinfo. 2023. *The trusted source for IP address data, leading IP data provider - IPinfo.io*. https://ipinfo.io (Accessed on 2023-09-04).
[17] Ipqualityscore. 2023. *Fraud Detection and Bot Detection Solutions | Detect Fraud with IPQS*. https://www.ipqualityscore.com (Accessed on 2023-09-04).
[18] Minghao Jiang, Zhen Li, Peipei Fu, Wei Cai, Mingxin Cui, Gang Xiong, and Gaopeng Gou. 2022. Accurate mobile-app fingerprinting using flow-level relationship with graph neural networks. *Computer Networks* 217 (2022), 109309.
[19] Mohammad Taha Khan, Joe DeBlasio, Geoffrey M Voelker, Alex C Snoeren, Chris Kanich, and Narseo Vallina-Rodriguez. 2018. An empirical analysis of the commercial vpn ecosystem. In *Proceedings of the Internet Measurement Conference 2018*. 443–456.
[20] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[21] Martin Laštovička, Martin Husák, Petr Velan, Tomáš Jirsík, and Pavel Čeleda. 2023. Passive operating system fingerprinting revisited: Evaluation and current challenges. *Computer Networks* 229 (2023), 109782.
[22] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. 2022. Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In *Proceedings of the ACM Web Conference 2022*. 633–642.
[23] Chang Liu, Longtao He, Gang Xiong, Zigang Cao, and Zhen Li. 2019. Fs-net: A flow sequence network for encrypted traffic classification. In *IEEE INFOCOM 2019-IEEE Conference On Computer Communications*. IEEE, 1171–1179.
[24] Efrén López-Morales, Carlos Rubio-Medrano, Adam Doupé, Yan Shoshitaishvili, Ruoyu Wang, Tiffany Bao, and Gail-Joon Ahn. 2020. Honeyplc: A next-generation honeypot for industrial control systems. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 279–291.
[25] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. 2020. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing* 24, 3 (2020), 1999–2012.
[26] Gordon Fyodor Lyon. 2009. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*.
[27] Aniss Maghsoudlou, Lukas Vermeulen, Ingmar Poese, and Oliver Gasser. 2023. Characterizing the VPN Ecosystem in the Wild. In *International Conference on Passive and Active Network Measurement*. Springer, 18–45.
[28] Shane Miller, Kevin Curran, and Tom Lunney. 2018. Multilayer perceptron neural network for detection of encrypted VPN network traffic. In *2018 International Conference On Cyber Situational Awareness, Data Analytics And Assessment (Cyber SA)*. IEEE, 1–8.
[29] Neo4j. 2023. Graph Algorithms. https://neo4j.com/docs/graph-data-science/current/algorithms/ (Accessed on 2023-09-04).
[30] Mohammad Saidur Rahman, Payap Sirinam, Nate Mathews, Kantha Girish Gangadhara, and Matthew Wright. 2019. Tik-Tok: The utility of packet timing in website fingerprinting attacks. *arXiv preprint arXiv:1902.06421* (2019).
[31] Reethika Ramesh, Leonid Evdokimov, Diwen Xue, and Roya Ensafi. 2022. VPNalyzer: systematic investigation of the VPN ecosystem. In *Network and Distributed System Security*. 24–28.
[32] Nhien Rust-Nguyen, Shruti Sharma, and Mark Stamp. 2023. Darknet traffic classification and adversarial attacks using machine learning. *Computers & Security* 127 (2023), 103098.
[33] Teemu Rytilahti and Thorsten Holz. 2020. On Using Application-Layer Middlebox Protocols for Peeking Behind NAT Gateways.. In *NDSS*.
[34] Danfeng Shan, Fengyuan Ren, Peng Cheng, Ran Shu, and Chuanxiong Guo. 2019. Observing and mitigating micro-burst traffic in data center networks. *IEEE/ACM Transactions on Networking* 28, 1 (2019), 98–111.
[35] Spur. 2023. *Home - Spur*. https://spur.us (Accessed on 2023-09-04).
[36] Pluggable Transports. 2023. *Software Repository*. https://software.pluggabletransports.info (Accessed on 2023-09-04).
[37] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).
[38] Vpnapi.io. 2023. *VPN and Proxy Detection API*. https://vpnapi.io (Accessed on 2023-09-04).
[39] Gaukas Wang, Jackson Sippe, Hai Chi, and Eric Wustrow. 2023. Chasing Shadows: A security analysis of the ShadowTLS proxy. In *In Free and Open Communications on the Internet*.
[40] Han Wang, Xiangyang Luo, and Yuchen Sun. 2020. An Obfs-based Tor Anonymous Communication Anline Identification Method. In *2020 6th International Conference on Big Data and Information Analytics (BigDIA)*. IEEE, 361–366.
[41] Liang Wang, Kevin P Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. 2015. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 57–69.
[42] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.
[43] Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin, and Eric Wustrow. 2023. How the Great Firewall of China detects and blocks fully encrypted traffic. In *32nd USENIX Security Symposium (USENIX Security 23)*. 2653–2670.
[44] Renjie Xie, Yixiao Wang, Jiahao Cao, Enhuan Dong, Mingwei Xu, Kun Sun, Qi Li, Licheng Shen, and Menghao Zhang. 2023. Rosetta: Enabling robust tls encrypted traffic classification in diverse network environments with tcp-aware traffic augmentation. In *Proceedings of the ACM Turing Award Celebration Conference-China 2023*. 131–132.
[45] Hongbo Xu, Shuhao Li, Zhenyu Cheng, Rui Qin, Jiang Xie, and Peishuai Sun. 2022. VT-GAT: A Novel VPN Encrypted Traffic Classification Model Based on Graph Attention Neural Network. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer, 437–456.
[46] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).
[47] Diwen Xue, Reethika Ramesh, Arham Jain, Michalis Kallitsis, J Alex Halderman, Jedidiah R Crandall, and Roya Ensafi. 2022. {OpenVPN} is open to {VPN} fingerprinting. In *31st USENIX Security Symposium (USENIX Security 22)*. 483–500.
[48] Ao Yu, Hui Yang, Kim Khoa Nguyen, Jie Zhang, and Mohamed Cheriet. 2020. Burst traffic scheduling for hybrid E/O switching DCN: An error feedback spiking neural network approach. *IEEE Transactions on Network and Service Management* 18, 1 (2020), 882–893.
[49] Yinbo Yu, Xing Li, Xue Leng, Libin Song, Kai Bu, Yan Chen, Jianfeng Yang, Liang Zhang, Kang Cheng, and Xin Xiao. 2018. Fault management in software-defined networking: A survey. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 349–392.
[50] Ruijie Zhao, Xianwen Deng, Yanhao Wang, Libo Chen, Ming Liu, Zhi Xue, and Yijun Wang. 2022. Flow sequence-based anonymity network traffic identification with residual graph convolutional networks. In *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.
[51] Zhuang Zou, Jingguo Ge, Hongbo Zheng, Yulei Wu, Chunjing Han, and Zhongjiang Yao. 2018. Encrypted traffic classification with a convolutional long short-term memory neural network. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 329–334.
[52] Zscaler. 2023. *Understanding PAC Files*. https://help.zscaler.com/zia/understanding-pac-file (Accessed on 2023-09-03).

## A PORT COMBINATION

We show examples of some VPNs' probing port combinations in Table 6. Some VPNs may have multiple port combinations. For instance, Psiphon3 has [443, 53, 22], [80, 554, 22], [80, 22], and other combinations. This variability could be due to firewall restrictions and the the complete set of port combinations may be [554, 22, 53, 80, 443]. Researchers can mitigate this issue by conducting probes using nodes in different geographical locations.

**Table 6: Example of Probing Port Combinations**

| VPN Name | Probing Port Combination |
|---|---|
| Psiphon3 | {22, 53, 443} |
| Psiphon3 | {22, 443, 554} |
| Psiphon3 | {22, 53, 443, 554} |
| Surfshark | {443, 1443, 4000, 7443, 8443} |
| PIA | {80, 443, 8443} |
| PIA | {80, 443, 8080, 8443, 8888} |
| Hotspot Shield | {80, 443, 563, 636, 993, 995} |
| Hotspot Shield | {80, 443, 563, 636, 993, 995, 6000} |
| CyberGhost | {443, 8080, 8081, 8443} |
| Easy | {80, 443, 8080, 8088, 12345} |
| Vpnify | {22} |
| HideMyAss | {80, 443} |
| Tomato | {22, 80, 443, 8000} |
| Touch | {443, 8082, 8083} |
| Windscribe | {443, 8443} |

**Table 7: VPNs in our dataset**

| VPN Name | | |
|---|---|---|
| AirVPN | Fastestvpn | ItopVPN |
| Browsec | Fastvpnio VPN | Just VPN |
| Cryptostorm | Foxyproxy | NordVPN |
| CyberGhost | Gecko | Opera |
| Daily | HideMyAss | PIA |
| Deeper Network | Hotspot Shield | Privatevpn |
| Easy | Infvpn | Proton |
| ExpressVPN | IC | Psiphon3 |
| Witopia | Xvpn | Zenmate |
| Pure | Trust Zone | Thunder |
| Secure Android | Tunnelbear | Tomato |
| Supervpn360 | Turbo | TorGuard |
| Surfshark | Ultrasurf | SuperVPN |
| Vpnify | Vpnunlimited | Windscribe |
| Touch | unknown | |

## B TOPOLOGICAL FEATURES

Here is a brief introduction to the topological features. **Degree**: The degree of a node in a graph is the number of edges incident to it. **Eigenvector Centrality**: It is used to express the importance of nodes, assuming that nodes connected to important nodes should also possess higher importance. **PageRank**: PageRank is an algorithm that calculates the ranking of nodes in a graph, originally designed for ranking web pages. **Closeness Centrality**: Closeness Centrality reflects the proximity of a node to all other nodes in the network. **Local Clustering Coefficient**: The local clustering coefficient is used to describe the connectivity between a node and its neighbors. **K-core**: The k-core of a graph is a maximal subgraph where every vertex has at least degree $k$, with $k$ reflecting the node's connectivity relative to other nodes in the network. The above node features are very common in graph analysis, and researchers can further extend the topological features to improve model performance.