IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received September 7, 2020, accepted October 26, 2020, date of publication November 16, 2020, date of current version December 1, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3038141

Detecting Malicious Users Behind Circuit-Based Anonymity Networks

SHOU-HSUAN STEPHEN HUANG, (Senior Member, IEEE), AND ZECHUN CAO, (Member, IEEE)
Department of Computer Science, University of Houston, Houston, TX 77204, USA

Corresponding author: Shou-Hsuan Stephen Huang (shuang@central.uh.edu)

This work was supported in part by the National Science Foundation under Grant NSF-1433817, Grant NSF-1356705, and Grant NSF-1950297; and in part by the U.S. Department of Homeland Security under Grant D15PC00185.

ABSTRACT This project addresses the issue of detecting intruders from hiding behind privacy-protecting anonymity networks. The freely available Tor and the SOCKS proxy services have been popular tools that provide circuit-based anonymous connections to network users. However, recent security breaches reveal that SSH and HTTPS have been used to launch attacks by malicious users by taking advantage of these services to hide their identities. This paper investigates strategies to detect SSH and HTTPS connections via the circuit-based anonymity networks, to help servers and websites defend against anonymous intruders. We evaluate our approaches with SSH and HTTPS connections and show that they achieve high performance for both applications. Our detection algorithms are based on the extra latency delays introduced by the presence of the anonymity networks. Since the latency disparity is sensitive to the location of the anonymity network, our algorithms must be evaluated in the most challenging scenarios. The detection rates for all four combinations of SSH/HTTPS applications via Tor/SOCKS networks were very high, with a low false-positive rate. To demonstrate the robustness of our approach in the Tor case, we tested our method in multiple Tor circuit node selection strategies. The approach can be applied to other applications meeting the same criteria.

INDEX TERMS Network security, intrusion detection, anonymous proxies, anonymity networks, circuit-based connections.

I. INTRODUCTION

This paper addresses the issue of detecting hackers from using privacy-protecting networks to hide their identity for malicious purpose. Internet users are facing an unprecedentedly high risk of being tracked and monitored. Technologies exist to assist online users in concealing their identities by using anonymous connections. However, the use of an anonymity network to protect a user's privacy comes with a cost. The extra hops will undoubtedly slow down the communication between the user and the server. While there may be legitimate reasons for people to use an anonymity network to protect their privacy to some servers or websites, there is no reason to do so for most of the servers in the world. For example, servers at the security-sensitive organizations, such as those of defense contractors, banks, hospitals, and research institutes, are designed for authorized personnel only. These servers require users to identify themselves using a secure connection (SSH or VPN). Therefore, it is very suspicious to have the user connected behind an anonymity network.

Since the global pandemic of Covid-19, there have been reports of economic espionage against many medical centers, universities, and pharmaceutical companies. Earlier this year, the UK government warned healthcare groups involved in response to the virus were being targeted by malicious actors [1]. Furthermore, hackers have used anonymity technologies to hide their identities in numerous attacks. Back in 2013, Target has reported data breach costs of $248 million incurred from a massive data breach, the largest in US history at the time [2]. In 2018, Facebook's security breach impacted almost 50 million user accounts, exposed everything in a victim's profile to the attackers. The US regulators have reportedly voted to fine Facebook $5 billion for this incident [3]. Although both breaches severely compromised a substantial amount of users' privacy, the hackers behind these attacks remain unidentified.

Among the existing anonymization technologies, routing traffic through one or more proxy servers is a flexible and reliable approach [4]. A proxy server establishes a connection

The associate editor coordinating the review of this manuscript and approving it for publication was Tiago Cruz.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

with a server on behalf of a client to conceal the client's real identity. Based on the way the proxy server routes the traffic, there are two types of proxy servers: application-level proxy and circuit-level proxy [5]. An application-level proxy only works for the particular application it is providing proxy service for, such as HTTP proxy. On the other hand, a circuit-level proxy creates a circuit between the client and the server to route traffic; therefore, it is more generic and can be adapted to serve almost any protocol [5]. Despite the circuit-level proxy's flexibility, it does not offer built-in encryption to its routing traffic, which makes profiling users' online activities possible. Additionally, locating a proxy server with high anonymity and consistent network performance is a non-trivial task.

However, the circuit-level proxy lays the foundation of several out-of-the-box anonymity networks such as the Tor network [6]. Unlike the original circuit-level proxy, which requires online users to locate and configure proxy servers in the circuit, the Tor network allows the users to take advantage of its pre-built high-anonymity circuit with multi-layered encryption. Since the Tor network relies on SOCKS protocol, which stands for Socket Secure, the same protocol supporting the circuit-level proxy anonymizes many applications over TCP connection like web browsing, secure shell (SSH), and instant messaging [6].

Not surprisingly, hackers have taken advantage of both proxy and Tor to conceal their identities. In 2012, Symantec researchers found that a piece of malware converts victim computers into a proxy network that funnels potentially malicious traffic through it [7]. Similarly, based on the data across the Cloudflare network, 94% of requests transmitted via the Tor network are malicious [8]. Recent research conducted by IBM also indicated that the Tor network is becoming a source of the malicious traffic that arises from Tor exit nodes [9]. Nonetheless, the state-of-the-art methods in detecting connections routed through the circuit-based anonymity network mainly based on the IP address blacklist, which may result in various issues, such as under-blocking and over-blocking. The blacklist of the rapidly changing proxy network contains only the proxies identified from past incidents or well-known proxy service providers. Even for the Tor network, whose exit nodes are publicly available, servers and websites often refer to an inaccurate or outdated copy of the blacklist or block a Tor exit node with a dynamic IP address. More serious issues may arise when "bottleneck" web services (e.g., CloudFlare, Akamai), whose components are used by many popular websites, block or discriminate against Tor users by the blacklist-based approaches [10]. Examples of the outcomes resulting from these issues include deterring people from contributing their computer as exit nodes, blocking non-Tor users from popular websites, and allowing Tor-originated abuse accessing critical web services.

Among many applications that can be exploited by hackers to cause damage to the victim server, SSH is widely available on virtually all Unix-inspired operating systems as well as Microsoft Windows, and it has become one of the most popular applications to administrate remote servers and transfer files [11]. Moreover, SSH tunneling provides even more applications for users to hide their network traffic or identities. These functionalities leave SSH host at a higher risk of being attacked by insecure remote connections [12], [13]. In 2011, Visa issued a security alert that states: "Insecure remote access, ranging from command-line based (SSH, Telnet) to visually-driven packages (pcAnywhere, VNC, MS Remote Desktop), continues to be the most frequent attack method used by intruders to gain access to a merchant's point-of-sale (POS) environment" [14]. Although we realize that some SSH users may use proxy to bypass the firewall, connecting to an SSH server via an anonymity network, such as Tor, are likely to have malicious intentions.

HTTPS protocol is widely used on the Internet, transmits information between the clients and the websites. As of April 2018, 57.1% of the Internet's most popular websites have a secure implementation of HTTPS, and 70% of page loads use HTTPS [15]. However, a security report by Cyren [16] substantiated that HTTPS is utilized in 37% of all malware, and distributes every major ransomware family at some point. Moreover, HTTPS has been reportedly abused by hackers for their political manipulation efforts. Recently, Facebook takes down many deceptive accounts and pages accused by the US authorities of interfering with the 2016 presidential election [17]. These fake accounts on Facebook are often created anonymously via HTTPS connections to create influence campaigns [18]. Therefore, an effective method to defend server and website against anonymous SSH and HTTPS connections is urgently needed.

Our work in this paper is based on the hypothesis that a connection via circuit-based anonymity networks exhibits different network behavior compared to a direct connection, which is discernible at the server. We present a connection-based server-side approach to detect the circuit-based anonymous connections and validate its performance in this paper. This paper expands our prior work on detecting malicious SSH users hiding behind the Tor [19] by including the detections for the SSH connections through SOCKS proxy and HTTPS connections through Tor and SOCKS proxy.

The contributions of this paper are as follows. This paper presents a per-connection based intrusion detection method to detect intruders hiding behind the Tor network and SOCKS proxy chains. We validate the detection method with our collected SSH and HTTPS connections on the Internet. The detection performance of our method is excellent with true positive rates in the high 90 percentages and a false-positive rate of less than 2%. The protocol ratio method used in this paper may work with other protocols. Our detection method works by monitoring the protocol's handshake process, thus can detect intruders before transmitting the first data packet.

We organize the rest of this paper as follows. In Section II, we introduce anonymous networks and recent research in detecting such networks. Section III discuss the protocols of SOCKS and Tor to establish the proxy chain. In Section IV, we present an abstract model which explains

the latency disparity in the connection network for both Tor and SOCKS. The discrepancy then leads to a detection algorithm. In Section V, experimentation setups and the result of the testing to validate our approach are given. Finally, we conclude the paper by presenting the contribution and limitation of this work and its impact on cybersecurity.

II. BACKGROUND AND RELATED WORKS

We divide the background and related works into two subsections: the existing methods to build anonymous connection, which may be abused by malicious users, and previous researches associated with detecting and blocking the malicious anonymous connections.

A. ANONYMOUS CONNECTION

The goal of establishing an anonymous connection is to allow users to communicate with a remote server while concealing their identities. However, anonymity may also be used by attackers to protect them. In the past, hackers often take advantage of several intermediate hops to hide their identities before launching an attack. One example of such attacks is the SSH stepping-stone attack, by which the attacker gains access to a remote server with previously compromised accounts. With a chain of stepping-stones, the attacker has established a secure connection to a server anonymously.

Following the design of the original message-based anonymity networks [20]–[24], the circuit-based proxy framework built on top of the SOCKS [25] protocol offers several ready-to-use, low-latency anonymity solutions. Although such anonymity networks are designed to protect the user's privacy, they can be abused by malicious users to launch an anonymous attack without compromising any of the intermediate hosts. In contrast to the application-level proxy framework, which is mainly studied and deployed in the organizational firewall application, the circuit-level proxy framework has very little control over what happens through the proxy [5], thus has been widely used for anonymizing various protocols' traffic. The circuit-level proxy provides the foundation of the implementation of many relay-based anonymity network projects, such as Tor [6], I2P [26], AN.ON [27], and other commercial proxy aggregators. Compared to Tor, the I2P network has limited out-proxy servers to the clearnet, and AN.ON has designated proxy servers that can be easily targeted by hackers or government officials. Therefore, Tor is the common tool for proxying the traffic to the clearnet with low-latency anonymity. To launch an attack via Tor, the attacker can route their traffic through a preconfigured circuit with a high anonymity level. Furthermore, continuous development and research [28]–[30] keep improving the performance and anonymity of the existing Tor framework.

B. DETECTING ANONYMOUS CONNECTIONS

As we stated, hackers have been widely abusing anonymity technologies to hide their identity. Previous research has studied stepping-stone detection techniques, and researchers propose various algorithms to detect incoming SSH connections through stepping-stones. Kuo *et al.* [31] proposed an algorithm based on association rule mining to detect stepping-stone SSH attacks. The algorithm achieves relatively high accuracy even with chaffed or jittered network traffic. Similarly, Wu *et al.* [32] design a neural network-based algorithm to detect stepping-stone attacks and prove that the method can even predict the number of stepping-stones in the chain. Ding *et al.* [12] propose a novel approach that can identify long stepping-stone connection chains from short chains by comparing connections to a pre-computed short-chain profile. However, these algorithms rely on the timing information extracted from a large number of SSH connections data packets, results in late detection that hinders the victim server from taking any effective security measure.

Among the efforts of detecting anonymous connections through the circuit-level proxy and the Tor network, the vast majority of the existing research requires taking control or monitoring the intermediate hops or the client. In [33], Bai *et al.* propose a fingerprinting method to identify the Tor network traffic. Their approach uses specific strings, packet length, and the timing information of the packets. The authors test their way on simulated networks obtaining higher than 95% detection accuracy for the simulated Tor network. Ling *et al.* [34] present an analysis of the Tor traffic using an Intrusion Detection System (IDS). The paper presents the results of a study done using Suricata, and a commercial IDS ruleset (ETPro). According to their findings, 10% of the Tor traffic is malicious and triggers an alert. From that 10%, P2P traffic triggers more than 70% of the warnings. Lashkari *et al.* [35] present a time analysis on the Tor traffic flows captured between the client and the entry node. The authors define two scenarios, one to detect the Tor traffic flows and the other to identify the application type, such as browsing, chat, streaming, mail, VoIP, P2P, or file transfer. By validating with their generated dataset, the paper shows that general machine learning algorithms, such as Random Forest, KNN, achieve good results with time features extracted from the Tor traffic flows. Despite the effectiveness in detecting the Tor traffic in the existing research mentioned above, it is challenging for a server to implement these methods because taking control or monitoring other parts of the Internet is unrealistic.

Current anonymous connections detection methods on the server-side fall into two categories, filtering IP addresses based on a blacklist and analyzing the patterns of data packets in the network traffic stream. Although blacklist-based methods are not ideal, researchers [36] explore a wide range of anonymous networks and propose a multi-tiered detection technique based on the attained knowledge from various sources. In the literature of analyzing patterns in the network stream, previous works attempt to solve various problems. Abdou *et al.* [37], [38] use packets latencies for geolocation verification and reliable one-way delay estimation. But their methods require additional devices deployed on the Internet. Aghaei-Foroushani *et al.* [39] use a machine

learning-based approach to detect proxy server by the traffic log file without any access to the proxy server or the clients behind it. By testing with Squid proxy server configurations, the authors claim a 94.1% true positive rate and a 7.9% false-positive rate for distinguishing proxy traffic from direct traffic. Similarly, Webb *et al.* [40] propose two methods of detecting proxies at the service: one for TCP and one for the application layer. Both methods employ timing information measured in the network traffic flow and achieve up to 99% detection rates with meager false-positive rates. Dhaka [41] *et al.* also proposed an approach for proxy detection at the application layer. While these existing works analyze timing information in network flow on the server-side, they require collecting a large amount of data packets in the traffic before the detection. It may be too late to raise the alarm once the data packets are exchanged

On the contrary, we study the timing characteristics of the packet exchanges in the protocol handshake process, thus mitigating the security threat results from transmitting data packets. More importantly, detecting anonymous connections based on the protocol handshake process allows our method to be able to perform in a real-time detection system. Before we formally introduce our detecting approach, in the next section, we provide the analysis of packet exchange in establishing connections through anonymity networks.

III. ESTABLISHING ANONYMOUS CONNECTION

Because our detection method focuses on the timing information during the connection's initiation process, in this section, we explain the proxy's handshake and Tor's circuit creation processes.

A. CONNECTING THROUGH PROXY

Proxies are widely used in both building Internet firewalls and anonymizing connections. In general, proxies can operate at either the application layer or the session layer of the OSI model. Specifically, we categorize the proxy server that works at the application layer as the application-level proxy, for instance, HTTP proxy. On the other hand, the circuit-level proxy operates at the session layer, also called the shim layer between the application and transport layers in the TCP/IP model. As a consequence, the advantage of a circuit-level proxy over an application-level one is that it provides service for a wide variety of different protocols in both TCP and UDP domains [5].

As the de facto standard protocol of the circuit-level proxies, SOCKS is an Internet protocol that exchanges network packets between a client and server through a proxy server. The first SOCKS protocol proposed in 1992 [25] later evolves into SOCKS4 and SOCKS5, two significant versions. SOCKS5 provides numerous authentication methods for the secured client-server connection, as well as the support for the UDP-based applications, such as DNS resolution [25]. Because of these advantages over SOCKS4, the SOCKS5 protocol is more prevalent in commercial anonymous proxy servers, which provide services for Internet users to ensure their online anonymities. Consequently, we focus on the proxies running on top of the SOCKS5 protocol in this paper. Since our method relies on the timing information of the exchanged packets during the handshake, we start by introducing the SOCKS protocol handshake process.

1) SOCKS PROTOCOL HANDSHAKE

A SOCKS proxy can be placed in the middle of the connection to conceal the client's identity to the server. After the SOCKS protocol handshake, the proxy forwards all the subsequent data packets on the client's behalf to the destination server.

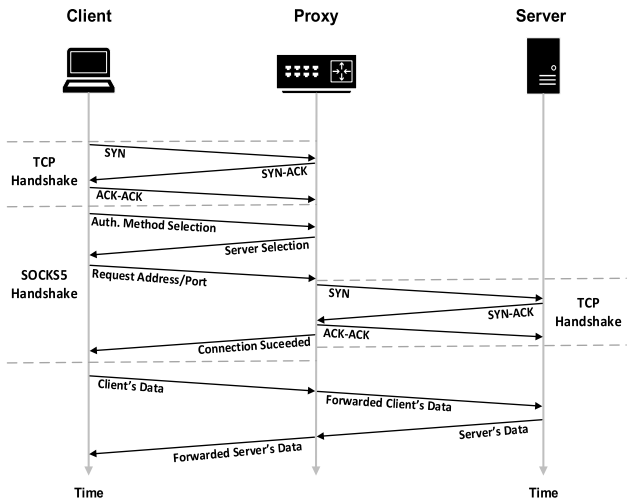


FIGURE 1. A depiction of how a client connect to a server via a SOCKS proxy server. Each arrow represent a packet transmission.

We depict an example in Figure 1, where a TCP-based client is connecting to a destination server via a SOCKS proxy. The connection starts with a TCP three-way handshake between the client and the proxy. Once they establish the TCP connection, the client sends the initiation packet that includes the version of the SOCKS protocol, the number of the authentication methods supported, and the list of the authentication methods to the proxy. The proxy then responds with its selection of the authentication method. After confirming the selected method, the client's next packet contains the destination server's IP address and the port number. Upon receiving the address information, the proxy initiates a TCP connection on behalf of the client and performs a three-way handshake with the destination server. Then the proxy notifies the client about the status of the TCP connection with the destination server. Granted that the connection succeeds, the client starts sending its application data, which is then masked by the proxy's IP address and forwarded to the destination server [25].

2) PROXY CHAIN THREAT MODEL

One of the principal objectives of an adversary attacking a remote server is to make identity tracing extremely difficult. With multiple proxies available, the attacker can chain them

together in the connection. The process of establishing a proxy chain is to perform the protocol handshakes incrementally from the client to the subsequent proxies in the chain. Additionally, different types of proxies, e.g., HTTP, SOCKS, can be mixed in the same chain, which offers more options for selecting proxies in the chain. Comparing to a one-proxy connection, tracking the client through a proxy chain is challenging because one would have to collect the log from each proxy. Moreover, it can take even more effort and time to trace back to the client if there are foreign proxies used in the chain.

However, configuring a private and anonymous chain of proxies can be complicated and time-consuming. One of the main factors that need the client's careful consideration is the connection speed. Because proxies in a chain might have different available bandwidths and latencies according to their configurations and network environments, the low-performance proxies can compromise the overall connection speed from the client to the destination. Besides, the proxy chain itself does not encrypt the network flows, results in possible eavesdropping along the chain. In other words, a client must trust every proxy in a proxy chain for a private and anonymous connection.

B. CONNECTING THROUGH TOR CIRCUIT

With the challenges of configuring a private and anonymous proxy chain in mind, Tor, a circuit-based low-latency anonymous network, provides the out-of-the-box anonymity service for the client. Similar to the proxy chain, Tor uses the standard and near-ubiquitous SOCKS proxy interface, allowing it to supports most TCP-based applications without modification [6]. To establish an anonymous connection to the client's intended destination, Tor chooses three suitable nodes among its volunteer-operated active relays. The three nodes, respectively, the entry, middle, and exit node, form a Tor circuit that forwards the network traffic on the client's behalf. A key aspect of achieving a low-latency connection while still maintaining a reasonable level of anonymity is that Tor's circuit selection algorithm selects nodes according to certain constraints. When building a circuit, Tor not only chooses nodes that are not in close network proximity but also weighs the nodes by their bandwidth, giving a higher selection probability to nodes with more bandwidth [6], [42].

Additionally, in Tor, Transport Layer Security (TLS) connections are used for the traffic encryption between two nodes to ensure the client's privacy. Except that the connection from the Tor exit node to the destination server is not encrypted, other connections along the circuit are all protected by TLS. That is, only the exit node in the circuit can examine the forwarded network traffic as clear messages. In Figure 2, we demonstrate the process of creating a three-node circuit for the TCP-based applications, which is a typical Tor configuration.

To anonymously connect to the destination server, the client first chooses a series of Tor nodes from the Tor directory, based on the circuit selection algorithm [6].

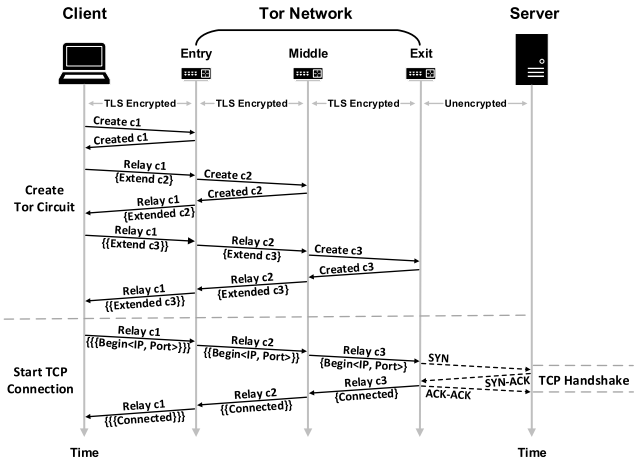


FIGURE 2. A depiction of how a client connect to a server via Tor network. The top portion shows the establishing of connection one at a time. The lower part of the figure show the TCP handshake.

In Figure 2, we denote the default three-node Tor circuit as an example. Similar to the process of chaining proxies, the client initiates the procedure of creating the Tor circuit incrementally, one hop at a time. We denote each segment between a pair of hops as a circuit segment. To create the first circuit segment *c1*, the client sets up a TLS connection with the entry node using the TLS protocol. Then, tunneling through this encrypted connection, the client sends a Tor command *"create"* to the entry node and negotiates a pair of symmetric keys to encrypt the traffic through c1 further. After the entry node responds with the *"created"* command, the circuit segment *c1* is successfully created.

To extend the *c1* one hop further, the client uses the negotiated key to encrypt and send a *"relay extend"* command to the entry node, specifying the address of the middle node. Once the entry node receives this command, it decrypts the message and sends a *"create"* command to the middle node to create the second segment *c2*. Identical to the process of creating *c1*, the entry and middle nodes negotiate a pair of symmetric keys used for the encryption in *c2*. Afterward, the middle node sends a *"created"* command, which indicates the success of building *c2* to the entry node. The entry node then notifies the client by sending an encrypted *"relay extended"* command. Subsequently, the client extends the last segment *c3* to the exit node by following the same procedure as in extending *c2*. As depicted in Figure 2, depending on the direction of the traffic flow, each Tor node in the circuit uses the negotiated symmetric keys to encrypt or decrypt the traffic in a layer by layer fashion.

With a complete Tor circuit, a TCP-based application can start an anonymous connection to a destination server by issuing a *"begin"* command, along with the destination server's IP address and the port number, to the Tor exit node. This three-layer encrypted command is then decrypted layer by layer while traversing through the circuit. After the exit node decrypts the last layer and learns the destination server's

identity, it initiates a TCP three-way handshake with the client's intended server.

It is worth noting that only the Tor exit node initiates the TCP connection with the destination server. In the TCP three-way handshake, the destination server learns the establishment of the TCP connection by receiving the ACK-ACK packet from the exit node. On the other hand, the client must wait until receiving the exit node's *"connected"* command to start sending its application data packet. Likewise, in the connection through a proxy chain, only the last proxy in the chain establishes the TCP connection with the server.

C. TOR CIRCUIT SELECTION STRATEGIES

Despite the constraints of the Tor's circuit selection algorithm, the Tor user can filter relays based on the locations. To select a customized circuit, the user configures a list of two-letter country codes in the Tor client. As a result, the selected Tor circuit contains only the relays that are not from the listed countries. Furthermore, the user can build a circuit that entirely resides in a single country by excluding all other countries' nodes. Therefore, we summarize three different circuit selection strategies that the Tor user can choose to build a Tor circuit [19].

Tor-Default (TD). Tor default circuit selection method is applied if there is no overriding circuit selection configuration. With thousands of active relays around the world, Tor often creates a circuit composed of geographically widespread relays.

Tor-Near-Client (TNC). We define the strategy of TNC as the client chooses the circuit that entirely resides in the client's located country. Although the lack of geographic diversity may reduce the level of anonymity, the TNC strategy is desirable if the user is in a censored network and unable to connect to the relays in other countries.

Tor-Near-Server (TNS). Opposite to the TNC, the TNS is the strategy that the user picks all relays geographically located within the same country as the server. For instance, a client in Germany can adopt the TNS strategy to create a US-based circuit to connect to a server in the US.

IV. METHODOLOGY

In this section, we describe our approach to detect circuit-based anonymous connections at the server-side. Our approach relies on the latency disparities within the protocol handshakes packets over the circuit-based anonymity networks. By analyzing the protocol handshake of the SSH and HTTPS, we find that routing through an extra anonymity network introduces latency disparities to the protocol handshake packets. These differences in latency provided us an opportunity of detecting connections routed through the anonymity networks. We start by explaining the latency disparity in the protocol handshake, which lays the foundation of our method. For the clarity of the notations in this section, we provide reader Table 1 for a summary of the defined timestamps.

TABLE 1. Server-side timestamp notation summary.

Protocol	Timestamp	Server's Event
TCP	$t_{SYN-ACK}$	Sending SYN-ACK
	$t_{ACK-ACK}$	Receiving ACK-ACK
SSH	t_{cp}	Receiving client's protocol announcement
	t_{sp}	Sending server's protocol announcement
	t_{ca}	Receiving client's algorithm negotiation
	t_{sa}	Sending server's algorithm negotiation
	t_{ck}	Receiving client's key exchange
	t_{sk}	Sending server's key exchange
HTTPS	t_{ch}	Receiving client's ClientHello
	t_{sh}	Sending server's ServerHello
	t_{cf}	Receiving client's Finished

A. NETWORK LATENCY DISPARITY MODEL

In the seven-layer Open Systems Interconnection (OSI) model, the protocol handshakes using a bottom-up approach. As a layer-4 transport protocol, TCP performs handshakes and sets up connections between each pair of the hosts in the path. After that, higher-layer protocols, such as SOCKS and Tor, provide an end-to-end channel for the client and server to communicate. Therefore, their protocol handshake packets exchange between the two ends of the channel along the entire path. After an end-to-end channel is established, layer-7 application protocols, such as SSH and HTTPS, utilize the channel and exchange the data on both ends.

Suppose a client connects directly to a server, without going through an anonymity network, all the protocol handshake packets exchange are between the client and the server. Therefore, there should not be a significant latency disparity. So the existence of substantial latency disparities in the protocol handshake packets is a strong indication of the presence of an anonymity network. We will demonstrate that this is indeed the case in Section V. In Figure 3, we depict our latency disparity model by assuming a three-node anonymity network structure. In the next section, we will elaborate on the analysis of the time gaps measured in the protocol handshakes of SSH and HTTPS. Then we introduce our approach based on the latency ratios computed by the time gaps.

In summary, our model assumes that the application of the user and the server must exchange information through the channel established. This model may be applied to other client-server applications if it also meets the requirement. It may worth pointing out that this assumption may not apply to some attack connections. If the attacker establishes a stepping–stone chain to hide the identity, the protocol information exchange occurs between each pair of nodes in the stepping-stone chain, but not end-to-end.

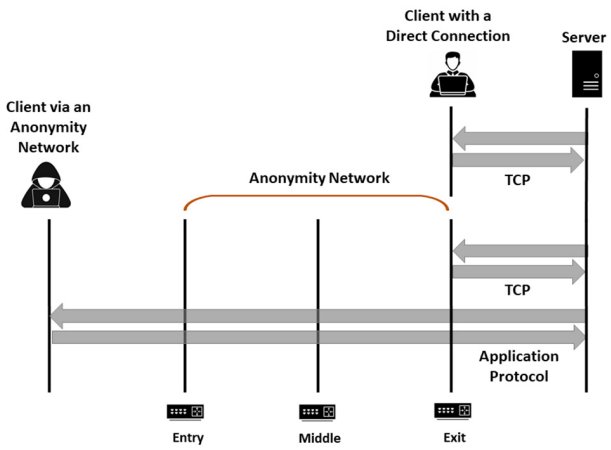
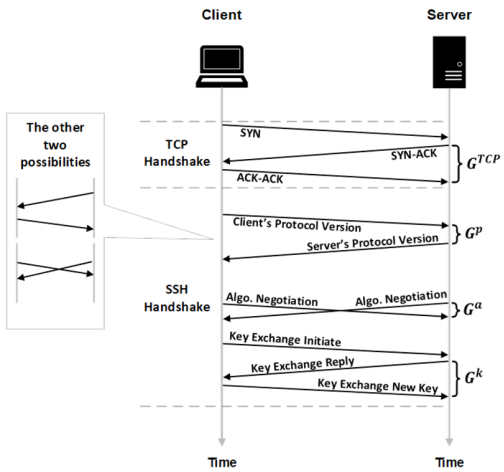
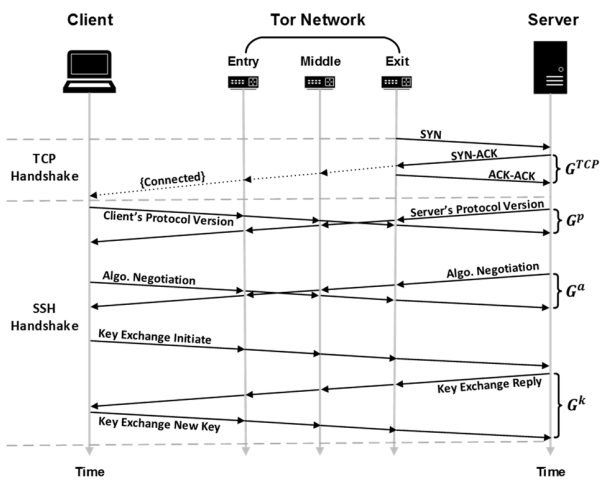


FIGURE 3. Network latency disparity in protocol handshakes.



(a) TCP and SSH handshakes in direct connection



(b) TCP and SSH handshakes via Tor network

FIGURE 4. Comparison of the TCP and SSH handshakes in direct connection and via the Tor network.

B. HANDSHAKE TIME GAPS IN SSH CONNECTION

Figure 4(a) depicts the handshake packets for TCP and SSH protocols in a direct SSH connecting scenario. In contrast, Figure 4(b) shows an SSH connection routing through a three-node Tor circuit. In both scenarios, we can estimate the TCP latency gap G^{TCP} at the server, by using two timestamps $t_{SYN-ACK}$, and $t_{ACK-ACK}$, we have:

$$G^{TCP} = t_{ACK-ACK} - t_{SYN-ACK}. \tag{1}$$

Because the server must wait for the last packet from the client to complete the TCP handshake process, G^{TCP} is considered a true round-trip time (RTT) [43]. Note that G^{TCP} only estimates the latency between the server and the last hop of the connection in both scenarios. Therefore, we use G^{TCP} as a measurement to compare with other time gaps.

Referring to the SSH protocol [44], the client and the server must negotiate and exchange necessary information before securely exchanging the data packets. The handshake starts with both sides announcing their protocol and software versions to ensure compatibility. As shown in Figure 4, there is only one packet from each side to complete this step. Since the SSH protocol [44] does not specify the order of sending these packets, either side can send it when it is ready. Therefore, it is possible that both packets crossover in the network. We compute the first handshake time gap G^p (p for protocol version) as:

$$G^p = t_{cp} - t_{sp}. \tag{2}$$

Using this definition, it is possible to have G^p as a negative value. In the direct SSH connection, the server likely sends its packet after receiving the client's packet. Then the negative G^p can provide a clear distinction with the connections through anonymity networks. Note that in the connection through the Tor circuit, both packets travel through the entire circuit.

The next step in the SSH handshake is to negotiate the algorithm for encryption key exchange, bulk data encryption, message integrity, and compression. Similarly, this step involves exchanging two packets, in which the client and server wrap a supported list of algorithms. As there is no order of packets defined based on the SSH protocol, each side should send the list after the previous handshake step completes successfully. Then G^a, the handshake time gap (a for algorithm negotiation) can be computed as:

$$G^a = t_{ca} - t_{sa}. \tag{3}$$

It is worth to mention that both G^p and G^a are not true RTTs, because the exchanging packets are not in a definite order.

The last step in the handshake is the Diffie-Hellman key exchange process, and it provides a shared secret that cannot be determined by either party alone [44]. In this step, we find that the number of involving packets varies among different software. But we identify three mandatory packets to be exchanged across implementations and depict them in Figure 4. The last SSH handshake time gap G^k (k for key exchange) is computed as:

$$G^k = t_{ck} - t_{sk}. \tag{4}$$

In contrast to the G^p and G^a, G^k is a true RTT due to the dependencies of the exchanging packets.

Within the SSH handshake process, we measure three handshake time gaps as G^i, $i \in \{p, a, k\}$. However, G^i alone does not reflect the latency discrepancy for the detection. Therefore, we take advantage of the latency G^{TCP} measured in the TCP three-way handshake, and compute the ratios of G^i against G^{TCP} as R^i, as in the following:

$$R^i = \frac{G^i}{G^{TCP}}, \quad i \in \{p, a, k\}. \tag{5}$$

For each incoming SSH connection, we can compute the R^i at the server, before the application data exchanging, and apply the detection algorithm with R^i to detect whether the connection is routed through a circuit-based anonymity network.

1) HANDSHAKE TIME GAPS IN HTTPS CONNECTION

Similar to Subsection B, we will identify the timestamp gaps for HTTPS here. Resides on layer seven as well, HTTPS is an extension of the HTTP protocol where communications between browsers and websites are encrypted. HTTPS uses Transport Layer Security (TLS), or its predecessor, Secure Sockets Layer (SSL), for secure end-to-end communication over the Internet. For this paper, we assume TLS protocol version 1.3 is used by the HTTPS.

As shown in Figure 5, the only step required to establish a TLS connection is to negotiate a cipher to encrypt the following network traffic. Two packets involved in this step are strictly ordered, i. e., the server has to wait until receiving the client's packet before sending out the next protocol packet. The TLS handshake concludes after the server's reply in most of the cases, and the client sends its Finished packet combined with the application data in its next packet. In this step, we introduce two handshake time gaps G^s and G^h (see Figure 5). G^s is computed in conjunction with the TCP handshake, as in below:

$$G^s = t_{ch} - t_{SYN-ACK}, \tag{6}$$

where G^s approximates an RTT between the client and the server. Based on the protocol, the second time gap G^h measures the true RTT as:

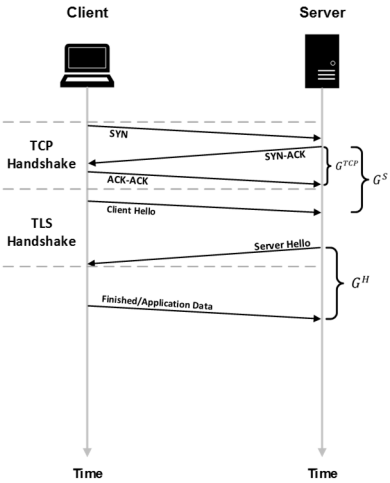
$$G^h = t_{cf} - t_{sh}. \tag{7}$$

In summary, we identify the two time-gaps as G^j, j $\in \{s, \mathrm{h}\}$ for the TLS handshake. Similar to the R^i defined for the SSH handshake, we then denote R^j for the HTTPS connections as in below:

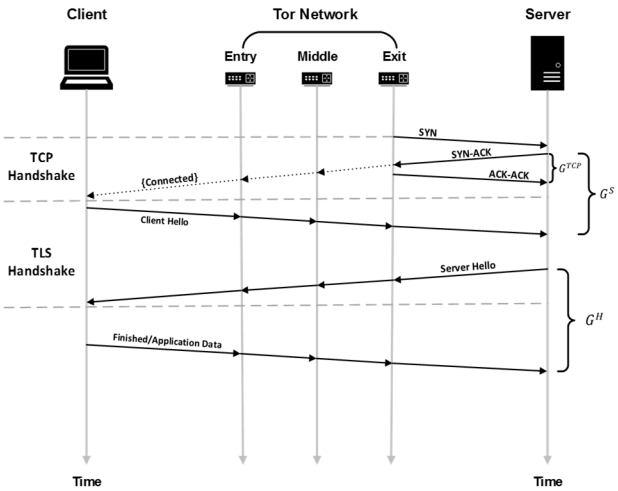
$$R^j = \frac{G^j}{G^{TCP}}, \quad j \in \{s, h\}. \tag{8}$$

Both R^i and R^j can reflect the latency discrepancy introduced by the extra hops in the connection.

After defining all these ratios, is there a latency disparity between a direct connection and one that goes through an anonymity network? In Figure 6, we plotted all the R^a values for both direct connection and connection via Tor.



(a) TCP and TLS handshakes in direct connection



(b) TCP and TLS handshakes via Tor network

FIGURE 5. Comparison of TCP and TLS handshakes in direct connection and via the Tor network.

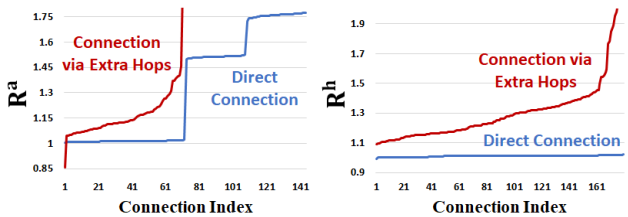


FIGURE 6. Sample latency disparities between direct connection and connection via extra hops in R^a for SSH and R^h for HTTPS.

The figure on the left shows the two sets of values sorted in increasing order. There is no intersection between the two sets of values. Similarly, we plotted the R^h values for the connections with and without the SOCKS proxy in sorted order on the right. It is evident that in both cases, there are significant disparities between the direct connections and the connections via extra hops. Although only ratios R^a and R^h

are shown in Figure 6, we observe the same latency disparity in all the ratios we defined. The experiment setup is discussed in Section V.

C. ANOMALY DETECTION ALGORITHM

There are two main approaches employed by intrusion detection techniques: misuse-based detection and anomaly detection. The misuse detecting approach defines abnormal behavior and relies on signature matching algorithms to detect anomaly [45]. However, this approach requires previous knowledge of the attacks, and even slight variants of known attacks often evade detection using signature matching algorithms. As a result, the misuse-based detecting approach cannot detect novel or unknown intrusions. On the contrary, the anomaly detection methods describe what is normal and identify the intruders by the deviations from the normal ones. Since it is infeasible to have labeled intruder's dataset in the training process, we choose to profile the normal cases—direct connections and identify the connection via the anonymity network as an anomaly. With the R^i and R^j computed, we can apply an anomaly detection algorithm to build a detection model for the SSH and HTTPS connections, respectively.

We adopt the isolation forest classifier [46] in a supervised setting by training the classifier with only the direct connections data. For the SSH connections, we can use any subset of $\{R^p, R^a, R^k\}$ as detecting features. We will explain the SSH attributes selected for the SSH connections in Section V together with their performances. For the HTTPS connections, we include both features R^s and R^h in the dataset. Next, we describe the algorithm assuming the dataset $D = (A_1, \ldots, A_m)$ is a multi-feature dataset.

In the training process, the classifier creates a forest of *iTrees*. To build an *iTree*, the classifier recursively divides the training dataset by randomly selecting a feature A_i in $D = (A_1, \ldots, A_m)$, and a split value in the range of A_i until either the terminating node has only one connection or all connections at the terminating node have the same A_i values. With a built *iTree*, the number of splittings required for each connection is equivalent to the path length from the root node to the terminating node. This path length averaged over a forest of such *iTrees* is used to define an anomaly score.

During the testing phase, the classifier finds the anomaly score for each connection in the testing dataset by navigating it through all *iTrees* in the forest. We expect the random partitioning produces noticeably shorter paths for the connections through a circuit-based anonymity network because they are different from the direct ones. When a forest of random trees collectively produces shorter path lengths for particular connections, they are highly likely to be the ones through an anonymity network. By varying the detecting threshold value in the range of anomaly scores, we can evaluate the classifier's performance in terms of AUC. We demonstrate our experiments' design and the evaluation results of detecting anonymous connection through circuit-based networks with a real-world collected dataset in Section V.

V. EXPERIMENTS AND RESULTS

In this section, we include the experiments and results of detecting the SSH and HTTPS connections via Tor and SOCKS proxy. In Subsection A, we start by explaining the setup of the experiments and our collected dataset. In Subsection B, we introduce our findings in a possible hacker's evasion from our detection method by carefully selecting the Tor circuit. Lastly, we demonstrate the high detection performance in detecting circuit-based anonymous connections in Subsection C.

A. EXPERIMENT SETUP

To simulate the SSH and HTTPS connections initiated from various locations on the AWS cloud, we use seven nodes that are geographically scattered by vast distances. We allocate each node in a different country and denote them by their two-letter country codes. Among the seven nodes, we configure two of them as servers, located in Sydney, Australia (AU), and Tokyo, Japan (JP). We then designate the rest of the nodes as clients. The locations for the five clients are Toronto, Canada (CA), Frankfurt, Germany (DE), Paris, France (FR), Stockholm, Sweden (SE), and Hilliard, Ohio (US). Consequently, we have ten different routes for the SSH and HTTPS connections, as each client can connect to a server as a unique path. We summarize the information about the selected hosts in Table 2. We collected our data to include a total number of 2160 SSH and HTTPS connections from ten different routes.

TABLE 2. Experiment participating hosts information.

Client/ Server	Location	OS	SSH Implementation	HTTPS Implementation
Client	Toronto, CA	Windows 10	Putty 0.67	cURL 7.65.0
Client	Frankfurt, DE	Ubuntu 14.04.01	OpenSSH 7.2	OpenSSL 1.1.1
Client	Paris, FR	Windows 10	Putty 0.67	cURL 7.65.0
Client	Stockholm, SE	Ubuntu 14.04.01	OpenSSH 7.2	OpenSSL 1.1.1
Client	Ohio, US	Ubuntu 14.04.01	OpenSSH 7.2	OpenSSL 1.1.1
Server	Sydney, AU	Ubuntu 14.04.01	OpenSSH 7.2	OpenSSL 1.1.1
Server	Tokyo, JP	Ubuntu 14.04.01	OpenSSH 7.2	OpenSSL 1.1.1

We deploy the seven nodes in the experiment on Amazon EC2 instances, each of which is equipped with a 1-core CPU and 1GB memory. All of our EC2 instances have Ubuntu 18.04 OS installed. To conduct the SSH connection experiments, we use the OpenSSH 8.2, which is natively and widely supported by Unix-like systems. For the HTTPS connections, we set up the TLS server with OpenSSL 1.1. The OpenSSL is a robust, commercial-grade, and full-featured toolkit for the TLS protocol, including the latest TLS 1.3. We implement the client's HTTPS request by the OpenSSL client tool.

Since our work focuses on Tor and proxy chain as circuit-based anonymity networks, we install both Tor client and proxy chain software on our nodes. The Tor client enables us to route the SSH and HTTPS connections through a live Tor circuit. To create a proxy chain from the node, we employ a software, ProxyChains, to redirect the SSH and HTTPS connections through a defined SOCKS5 proxy chain. To record and analyze the network packets, we collect the incoming packets at the server using Tshark, a terminal-based Wireshark toolkit. We capture each inbound stream of packets and save as a *pcap* file for further analysis.

B. HACKER's EVASION BY TOR CIRCUIT SELECTION

To evaluate our detection method's impact by adopting different Tor circuit selection strategies, we conduct a hacker evasion experiment on detecting the SSH connection through Tor. The dataset used in this experiment composed of 576 SSH connections collected from four different connection types. There are four connection types in this dataset, direct, TD, TNC, and TNS. For each of the SSH connections, we extract the set of features R^i, as described in Section IV. We plot $R^p \in R^i$ of four different connection types in Figure 7. We sort the 144 R^p values of each connection type are in ascending order to depict their relative relations. Note that the connection index in Figure 7 denotes each connection's label.

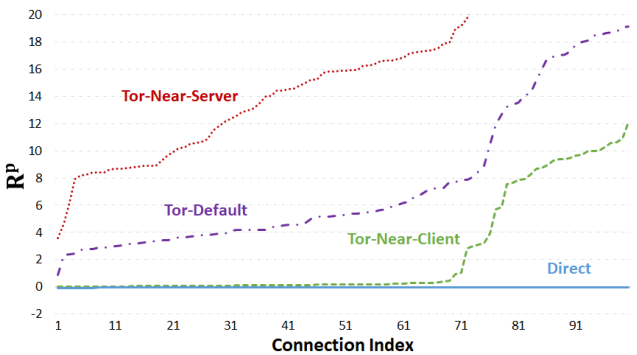


FIGURE 7. Comparison of R^p values in four connection types. It is very difficult to separate the R^p values in Tor-Near-Client (TNC) connections and the direct connections.

In Figure 7, note that R^p in the direct SSH connections has the lowest values, most are negative, on the graph, while TNC, TD, and TNS are clearly showing higher R^p values with noticeable gaps in between. Although not plotted, R^a and R^k are showing the identical pattern with R^p with the four connection types. It is interesting to observe that the direct connection and TNC have the smallest gap between them than the other curves. Without a closer observation of both curves' lower ranges, we find it difficult to separate them by a single cutoff threshold value. In other words, it is the most challenging task to detect the SSH connections through Tor from the direct ones if the Tor nodes are near the client (TNC). Similarly, the hackers may deliberately select the SOCKS proxies geographically closer to them to evade our detection.

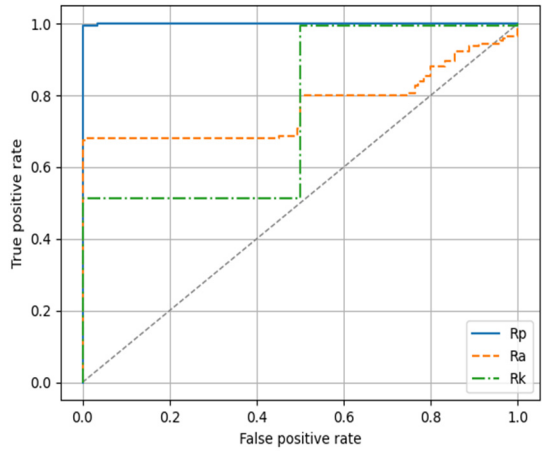


FIGURE 8. SSH features R^p, R^a, and R^k evaluations by ROC curves in detecting Tor connections with TNC strategy and direct connections. It shows that only R^p performs well under the worst possible scenario.

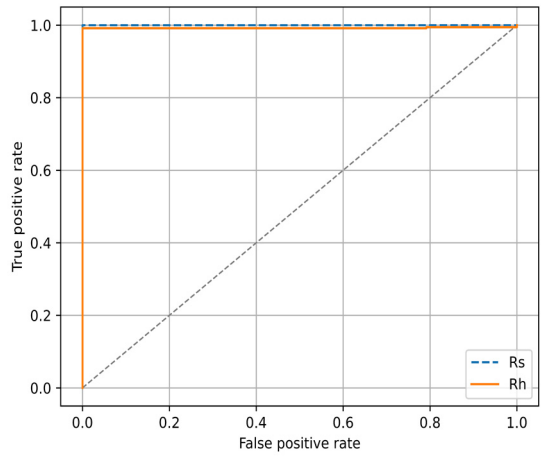


FIGURE 9. HTTPS features R^s, and R^h evaluations by ROC curves in detecting Tor connections with TNC strategy and direct connections. It shows that both R^s, and R^h perform well under the worst possible scenario.

As the ROC curves plotted in Figure 8, we test the effectiveness of R^p, R^a, and R^k under the worst possible scenario. We focus on the TNC strategy and extract R^i from the protocol handshakes. We then evaluate the individual performance of each metric by plotting its ROC curve in Figure 8. With a 0% false-positive rate, R^p achieves a very high true positive rate (TPR) of 99%, but R^a and R^k merely obtain 68% and 51%, respectively. Consequently, we decide to use R^p in the detection method for the SSH evaluations. Similarly, we test the effectiveness of the HTTPS features R^s and R^h in detecting the connections through Tor if the attacker adopts the TNC strategy. As the ROC curves plotted in Figure 9, both features perform very well even under the worst possible scenario. Thus, we use both R^s and R^h in our HTTPS evaluations.

C. EVALUATIONS

We evaluate our detection approach with the SSH and HTTPS connections, which route through Tor or the SOCKS5

proxy chain. For the connections through the proxy chain, reducing the number of hops diminishes the protocol handshake delays, resulting in less discernable detection features. Therefore, to evaluate our method with the most challenging cases, we use the one-proxy setting to connect through the proxy chain. Next, we evaluate our method in detecting the circuit-based anonymous SSH and HTTPS connections separately.

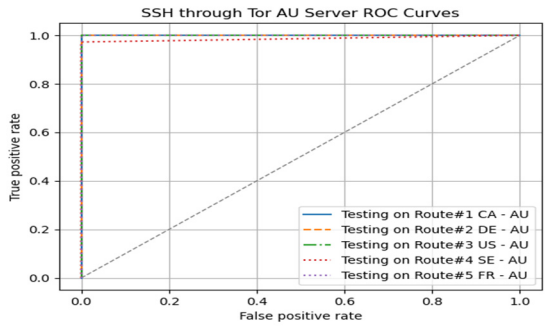
1) DETECTING SSH CONNECTIONS THROUGH CIRCUIT-BASED ANONYMITY NETWORKS

We conduct two experiments to evaluate our method with SSH connections. The first one is to evaluate the performance of identifying the SSH connections through Tor from the direct connections. On the other hand, the second experiment focuses on identifying the SSH connections through a SOCKS5 proxy from the direct connections. The dataset we use includes 1080 simulated SSH connections, in which there are 360 direct connections, 360 connections through Tor with TNC strategy, and 360 connections through a SOCKS proxy.

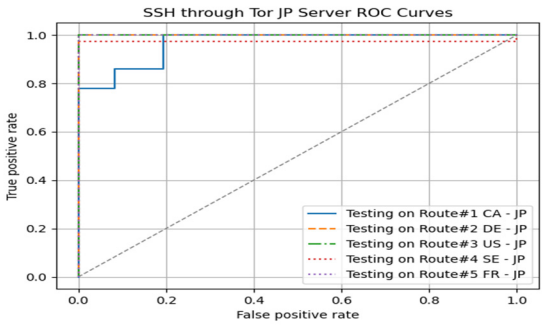
To evaluate our detection method, we divide the dataset into two groups, and each group contains the connections that all destinate to the same server. Then, we test our detection method separately on two groups and report the performance, respectively. We perform the five-fold cross-validation based on the five different routes to evaluate our method in each group of connections. By adopting the route-based grouped cross-validation, we ensure that the same route's connections do not appear in both testing and training phases. Therefore, the anomaly detector does not have any prior knowledge of the testing dataset.

In Figure 10, we depict the ROC curves of detecting the SSH connections through Tor on the AU server and the JP server, respectively. As shown in Figure 10, for each server, we depict the ROC curve of using each connection route as the testing set. The results show that our detection method performs better on the AU server than the JP server. For the AU server, five ROC curves have 99.44% of the TPR, while maintaining a 0% false-positive rate (FPR) on average. Although the performance on the JP server also reaches 99.44% of the TPR, however, the FPR increases to 3.89% on average.

For the experiment of detecting the SSH connections through a proxy, we plot the ROC curves for each server in Figure 11. Based on the results, we find that our detection method performs very well in detecting the SSH connection through a proxy. On the AU server, the average TPR is 99.44%, while the average FPR is at 0%. On the JP server, we observe a slightly lower TPR of 98.89% on average, with a 0% FPR.

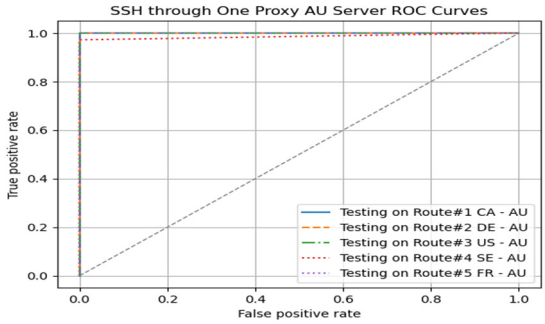


(a) Connections from 5 clients to the AU server

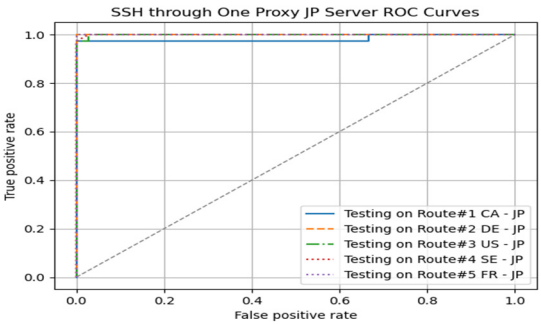


(b) Connections from 5 clients to the JP server

FIGURE 10. ROC curves of detecting SSH connections through Tor on two different servers.



(a) Connections from 5 clients to the AU server

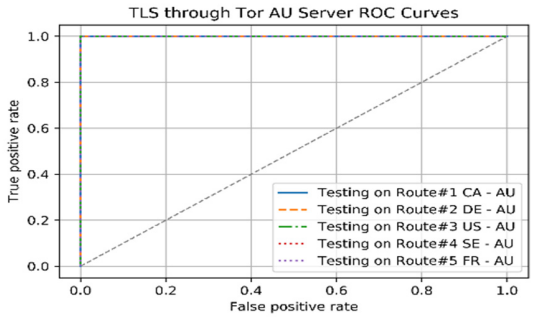


(b) Connections from 5 clients to the JP server

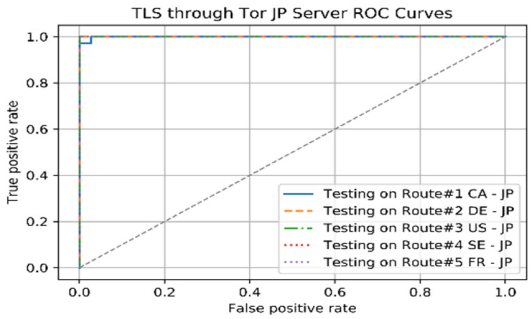
FIGURE 11. ROC curves of detecting SSH connections through a proxy on two different servers.

2) DETECTING HTTPS CONNECTIONS THROUGH CIRCUIT-BASED ANONYMITY NETWORKS

Similarly, by using the 1080 simulated HTTPS connections in the dataset, we conduct two experiments for the HTTPS detection, focus on detecting the connections through Tor and proxy, respectively. We plot the ROC curves for each experiment, as in Figures 12 and 13.

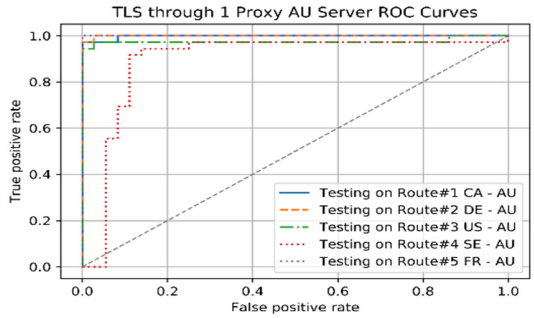


(a) Connections from 5 clients to the AU server

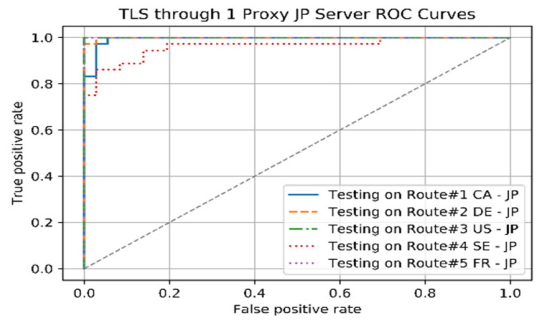


(b) Connections from 5 clients to the JP server

FIGURE 12. ROC curves of detecting HTTPS connections through Tor on two different servers.



(a) Connections from 5 clients to the AU server



(b) Connections from 5 clients to the JP server

FIGURE 13. ROC curves of detecting HTTPS connections through a proxy on two different servers.

In the experiment of detecting HTTPS through Tor, we achieved a 100% TPR with 0% FPR on the AU server. On the JP server, the average TPR is at 99.44%, with 0% FPR. The experiment of detecting HTTPS through one proxy shows a slightly lower overall performance.

TABLE 3. Performance evaluation for SSH and HTTPS.

Protocol	Anonymity Network	TP Rate (%)	FP Rate (%)	F-score
SSH	Tor	99.44	1.95	0.98
	Proxy	99.17	0	0.98
HTTPS	Tor	99.72	0	0.99
	Proxy	96.11	1.67	0.96

We obtained a 96.11% TPR with 2.22% FPR on the AU server and 96.11% TPR with 1.11% FPR on the JP server on average. Note that the TPR of detecting SSH and HTTPS connections through the proxy is lower than Tor. Since the proxy routes traffic without encryption, the latencies in protocol handshakes are lower than Tor, making it more challenging to be detected. Based on the experiments' results, we summarize our method's performance in detecting the SSH and HTTPS connections through Tor and proxy in Table 3.

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed a method to detect anonymous connections to a server through two types of circuit-based networks, Tor, and SOCKS proxy. Our method works by monitoring the latency disparities of network packets exchanged during the protocol handshakes. By conducting experiments on the Internet, we show that our method has achieved very high performance in detecting anonymous SSH and HTTPS connections. The contribution of this paper is significant because our approach can prevent malicious users from launching attacks anonymously, such as accessing security-sensitive servers and registering deceptive accounts on social media websites.

There are three features worth noting of our detection method, which makes it more robust. Firstly, we find that not all of the protocol handshake packets must be sent in a particular order. Some implementations of the protocol allow both sides to send out a packet whenever it is ready, in which case the time gap can be less than the true round-trip time. In the extreme case, the time gap value may even be negative, such as G^p measured in the SSH handshake. However, our method performs well regardless of the protocol timing. Secondly, we discover that the hackers may deliberately choose the anonymity network circuit that is geographically closer to them for detection evasion. Our approach works even under such challenging conditions possible. Lastly, our detection method uses only the information from the protocol handshake process, which is before the application data exchanges. Therefore, we can detect anonymous connections early to avoid potential threats. Furthermore, user behavior, such as typing speed and issued commands, does not impact the performance of our approach.

We note that there are several potential challenges posed by hacker's evasion techniques. Hackers can manipulate the timing of sending protocol packets to make our algorithm fail to detect the intrusion. For example, we have noticed that malicious Tor users could deliberately send out the pro-

tocol announcement packet in the SSH handshake sooner than it is supposed to be. Although we have not found any incident using the approach, it is theoretically possible. We need to study if our method is resilient enough for such manipulations.

REFERENCES

[1] BBC News. *Coronavirus: Cyber-Attacks Hit Hospital Construction Companies*. Accessed: May 13, 2020. [Online]. Available: https://www.bbc.com/news/technology-52646808
[2] N. E. Weiss and R. S. Miller, "The target and other financial data breaches: Frequently asked questions," in *Proc. Congressional Res. Service, Prepared Members Committees Congr.*, vol. 4, Feb. 2015.
[3] L. Matsakis and I. Lapowsky. (Sep. 2018). *Everything we Know About Facebook's Massive Security Breach*. Wired. Accessed: Oct. 3, 2020. [Online]. Available: https://www.wired.com/story/facebook-security-breach-50-million-accounts/
[4] J. McKeague and K. Curran, "Detecting the use of anonymous proxies," *Int. J. Digit. Crime Forensics*, vol. 10, no. 2, pp. 74–94, Apr. 2018, doi: 10.4018/IJDCF.2018040105.
[5] E. D. Zwicky, S. Cooper, and D. B. Chapman, *Building Internet Firewalls: Internet and Web Security*, 2nd ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2000.
[6] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *Proc. 13th Conf. USENIX Secur. Symp.*, vol. 13. San Jose, CA, USA: USENIX Association, 2004.
[7] B. Prince. (2012). *Symantec: Russian Hacker Behind Proxybox Trojan*. Accessed: Nov. 3, 2020. [Online]. Available: https://www.securityweek.com/symantec-russian-hacker-behind-proxybox-trojan
[8] M. Prince. (2016). *The Trouble With Tor*. Accessed: Nov. 3, 2020. [Online]. Available: https://blog.cloudflare.com/the-trouble-with-tor/
[9] *IBM X-Force Threat Intelligence Quarterly, 3Q 2015*, IBM Secur., Armonk, NY, USA, 2015, pp. 1–20.
[10] S. Khattak, D. Fifield, S. Afroz, M. Javed, S. Sundaresan, V. Paxson, S. J. Murdoch, and D. McCoy, "Do you see what i see? Differential treatment of anonymous users," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 21–24, doi: 10.14722/ndss.2016.23342.
[11] D. J. Barrett and R. Silverman, "The SSH protocol," in *SSH, The Secure Shell–The Definitive Guide*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2001, pp. 3–5.
[12] W. Ding and S.-H.-S. Huang, "Detecting intruders using a long connection chain to connect to a host," in *Proc. IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Mar. 2011, pp. 121–128, doi: 10.1109/AINA.2011.109.
[13] W. Ding, K. Le, and S.-H.-S. Huang, "Detecting stepping-stones under the influence of packet jittering," in *Proc. 9th Int. Conf. Inf. Assurance Secur. (IAS)*, Dec. 2013, pp. 31–36, doi: 10.1109/ISIAS.2013.6947729.
[14] *Visa Security Alert–Insecure Remote Access and User Credential Management*, VISA Inc., San Francisco, CA, USA, Jul. 2014, pp. 1–3.
[15] LetsEncrypt.org. (2018). *Let's Encrypt Stats*. Accessed: Nov. 3, 2020. [Online]. Available: https://letsencrypt.org/stats/
[16] A. Magnúsardóttir. (2017). *Malware is Moving Heavily to HTTPS*. Cyren Secur. Blog. Accessed: Mar. 12, 2020. [Online]. Available: https://www.cyren.com/blog/articles/over-one-third-of-malware-uses-https
[17] S. E. Needleman. (2020). *Facebook removes accounts tied to Russia's Internet Research Agency*. Accessed: Sep. 2, 2020. [Online]. Available: https://www.wsj.com/articles/facebook-removes-accounts-tied-to-russias-internet-research-agency-11599001139
[18] N. Fandos and K. Roose, "Facebook identifies an active political influence campaign using fake accounts," The New York Times, Jul. 2018.
[19] Z. Cao and S.-H.-S. Huang, "Detecting intruders and preventing hackers from evasion by tor circuit selection," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./12th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2018, pp. 475–480, doi: 10.1109/TrustCom/BigDataSE.2018.00074.
[20] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Commun. ACM*, vol. 24, no. 2, pp. 84–90, Feb. 1981, doi: 10.1145/358549.358563.
[21] U. Moeller, L. Cottrell, P. Palfrader, and L. Sassaman, "IETF draft: Mixmaster protocol version 2," 2004.
[22] G. Danezis, R. Dingledine, and N. Mathewson, "Mixminion: Design of a type III anonymous remailer protocol," in *Proc. 19th Int. Conf. Data Eng.*, May 2003, pp. 2–15, doi: 10.1109/SECPRI.2003.1199323.
[23] D. Kesdogan, J. Egner, and R. Büschkes, "Stop-and-go-MIXes providing probabilistic anonymity in an open system," in *Information Hiding*. Berlin, Germany: Springer, 1998, pp. 83–98.
[24] C. Díaz and A. Serjantov, "Generalising mixes," in *Privacy Enhancing Technologies*. Berlin, Germany: Springer, 2003, pp. 18–31.
[25] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and J. Jones, *SOCKS Protocol Version 5*. IETF. document RFC 1928, 1996.
[26] Jrandom. *The Invisible Internet Project (I2P)*. Accessed: Mar. 12, 2020. [Online]. Available: https://geti2p.net/
[27] H. Federath and S. Kopsell. *JAP: Java Anonymous Proxy*. Accessed: Dec. 3, 2020. [Online]. Available: http://anon.inf.tu-dresden.de/
[28] R. Annessi and M. Schmiedecker, "NavigaTor: Finding faster paths to anonymity," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Mar. 2016, pp. 214–226, doi: 10.1109/EuroSP.2016.26.
[29] M. Akhoondi, C. Yu, and H. V. Madhyastha, "LASTor: A low-latency AS-aware tor client," *IEEE/ACM Trans. Netw.*, vol. 22, no. 6, pp. 1742–1755, Dec. 2014, doi: 10.1109/TNET.2013.2291242.
[30] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal, "RAPTOR: Routing attacks on privacy in Tor," in *Proc. 24th Conf. USENIX Secur. Symp.* San Jose, CA, USA: USENIX Association, 2015, pp. 271–286. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/sun
[31] Y.-W. Kuo and S.-H.-S. Huang, "Detecting stepping-stone connection using association rule mining," in *Proc. Int. Conf. Availability, Rel. Secur.*, Mar. 2009, pp. 90–97, doi: 10.1109/ARES.2009.101.
[32] H.-C. Wu and S.-H. S. Huang, "Stepping-Stone intrusion detection using neural networks approach," in *Novel Algorithms and Techniques In Telecommunications, Automation and Industrial Electronics*. Dordrecht, The Netherlands: Springer, 2008, pp. 358–363.
[33] X. Bai, Y. Zhang, and X. Niu, "Traffic identification of tor and Web-mix," in *Proc. 8th Int. Conf. Intell. Syst. Design Appl.*, Nov. 2008, pp. 548–551, doi: 10.1109/ISDA.2008.209.
[34] Z. Ling, J. Luo, K. Wu, W. Yu, and X. Fu, "TorWard: Discovery of malicious traffic over tor," in *Proc. IEEE INFOCOM - IEEE Conf. Comput. Commun.*, Apr. 2014, pp. 1402–1410, doi: 10.1109/INFOCOM.2014.6848074.
[35] A. Habibi Lashkari, G. Draper Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features," in *Proc. 3rd Int. Conf. Inf. Syst. Secur. Privacy*, 2017, pp. 253–262, doi: 10.5220/0006105602530262.
[36] M. Pannu, B. Gill, R. Bird, K. Yang, and B. Farrel, "Exploring proxy detection methodology," in *Proc. IEEE Int. Conf. Cybercrime Comput. Forensic (ICCCF)*, Jun. 2016, pp. 1–6, doi: 10.1109/ICCCF.2016.7740438.
[37] A. Abdou, A. Matrawy, and P. C. van Oorschot, "CPV: delay-based location verification for the Internet," *IEEE Trans. Dependable Secure Comput.*, vol. 14, no. 2, pp. 130–144, Mar. 2017, doi: 10.1109/TDSC.2015.2451614.
[38] A. Abdou, A. Matrawy, and P. C. van Oorschot, "Accurate one-way delay estimation with reduced client trustworthiness," *IEEE Commun. Lett.*, vol. 19, no. 5, pp. 735–738, May 2015, doi: 10.1109/LCOMM.2015.2411591.
[39] V. Aghaei-Foroushani and A. N. Zincir-Heywood, "A proxy identifier based on patterns in traffic flows," in *Proc. IEEE 16th Int. Symp. High Assurance Syst. Eng.*, Jan. 2015, pp. 118–125, doi: 10.1109/HASE.2015.26.
[40] A. T. Webb and A. L. Narasima Reddy, "Finding proxy users at the service using anomaly detection," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2016, pp. 82–90, doi: 10.1109/CNS.2016.7860473.
[41] V. S. Dhaka, V. Lamba, Anubhav, and D. Pathania, "Application layer proxy detection, prevention with predicted load optimization," in *Proc. Int. Conf. Recent Adv. Innov. Eng. (ICRAIE)*, Dec. 2016, pp. 1–7, doi: 10.1109/ICRAIE.2016.7939513.
[42] R. Dingledine and N. Mathewson. (2008). *Tor Path Specification*. [Online]. Available: https://gitweb.torproject.org/torspec.git/plain/path-spec.txt.
[43] J. Postel, *Transmission Control Protocol*, Standard 7, RFC 793, Sep. 1981, [Online]. Available: https://www.rfc-editor.org/info/rfc793, doi: 10.17487/RFC0793.
[44] T. Ylonen and E. C. Lonvick, *The Secure Shell (SSH) Transport Layer Protocol*, document RFC 4253, Jan. 2006. [Online]. Available: https://www.rfc-editor.org/info/rfc4253, doi: 10.17487/RFC4253.
[45] S. Zanero and G. Serazzi, "Unsupervised learning algorithms for intrusion detection," in *Proc. IEEE Netw. Operations Manage. Symp. (NOMS)*, Apr. 2008, pp. 1043–1048, doi: 10.1109/NOMS.2008.4575276.
[46] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. 8th IEEE Int. Conf. Data Mining*, Dec. 2008, pp. 413–422, doi: 10.1109/ICDM.2008.17.