Cactus: Obfuscating Bidirectional Encrypted TCP Traffic at Client Side

Renjie Xie, Jiahao Cao, *Member, IEEE*, Yuxi Zhu, Yixiang Zhang, Yi He, Hanyi Peng, Yixiao Wang, Mingwei Xu, Kun Sun, *Member, IEEE*, Enhuan Dong, *Member, IEEE*, Qi Li, *Senior Member, IEEE*, Menghao Zhang, *Member, IEEE*, and Jiang Li

***Abstract*— As the mainstream encrypted protocols adopt TCP protocol to ensure lossless data transmissions, the privacy of encrypted TCP traffic becomes a significant focus for adversaries. They can leverage Deep Learning (DL) models to infer the sensitive information from encrypted TCP traffic by analyzing its packet size, direction, and timing information. To defend against such DL-based traffic analysis attacks, recent advances reshape the encrypted traffic and achieve desired results. However, they typically require deploying cooperative modules on both communication endpoints and only support specific applications, such as browsers. In this paper, we propose Cactus, a client-side plug-in to obfuscate bidirectional encrypted TCP traffic for a wide range of applications transparently using the inherent TCP semantics and the emerging eBPF technique. In particular, Cactus provides four effective operations to enable bidirectional traffic obfuscation while preserving communication semantics of applications. Besides, Cactus empowers users to specify which applications to conduct traffic obfuscation and what obfuscation level for each application. We conduct comprehensive experiments to demonstrate that Cactus can effectively obfuscate encrypted TCP traffic with low overhead to hinder the traffic analysis efforts in website fingerprinting and application identification.**

***Index Terms*— Encrypted TCP traffic, traffic analysis attacks, traffic obfuscation.**

I. Introduction

THE Transmission Control Protocol (TCP) is the dominant protocol in today's Internet, which provides reliable, ordered, and error-checked stream delivery for network applications. According to a large-scale investigation [32], more than 85% of total Internet traffic is TCP traffic. Meanwhile, to provide the privacy and anonymity for TCP traffic, various encryption protocols are built on top of it, such as Secure Sockets Layer (SSL) [25], Transport Layer Security (TLS) [19], [47], Secure Shell (SSH) [65], tcpcrypt [7], and Hypertext Transfer Protocol Secure (HTTPS) [46].

Despite the use of encryption, the privacy of encrypted TCP traffic remains at risk. Recent studies [21], [37], [48], [53] have shown that metadata including packet size, packet direction, and packet timing can still leak rich information about the traffic, raising significant privacy concerns. Particularly, the recent advances in deep learning offer powerful tools for adversaries to automatically and accurately infer sensitive information from the encrypted traffic. They can conduct various traffic analysis attacks, including application identification [5], [60], website fingerprinting [53], [54], user profiling [17], [23], operating system identification [30], [57], IoT device identification [2], [38], content identification of VoIP calls [4], [16], password guessing of secure shell logins [56], and even medical records and financial data identification [14].

To defeat these traffic analysis attacks and protect the traffic privacy, numerous defense approaches have been proposed to obfuscate the underlying traffic patterns [8], [9], [13], [21], [22], [29], [40], [41], [43], [44], [61], [62]. They typically modify the software or employ proxies tailored to specific applications at the endpoints to enforce various traffic obfuscation operations [8], [9], [21], [22], [29], [40], [43], [61], [62], such as packet padding, packet splitting, and dummy packet generation. Besides, recent studies [13], [41], [44] propose to leverage specific high-performance network devices, such as programmable switches, to add packet padding and introduce chaff packets into the traffic. Hence, the privacy for a large amount of network traffic can be efficiently safeguarded.

Although existing solutions can effectively obfuscate TCP traffic patterns, their practical deployment suffers from quite a few problems. First, they typically require two-side deployment to obfuscate the bidirectional flows to and from the client [8], [9], [13], [21], [29], [41], [43], [44], [61], [62]. In this way, the obfuscated flows can be correctly decoded without affecting the original communication semantics. However, modifying the remote endpoints or deploying network devices on the remote networks is out of the client's

Manuscript received 2 April 2024; revised 4 July 2024; accepted 23 July 2024. Date of publication 13 August 2024; date of current version 22 August 2024. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62132011, Grant 623B2062, Grant 62202260, and Grant 62221003. The associate editor coordinating the review of this article and approving it for publication was Dr. Guowen Xu. *(Corresponding authors: Jiahao Cao; Mingwei Xu.)*

Renjie Xie, Jiahao Cao, Yi He, Mingwei Xu, Enhuan Dong, and Qi Li are with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China (e-mail: xrj21@mailstsinghua.edu.cn; caojh2021@tsinghua.edu.cn; heyi21@mailstsinghua.edu.cn; xmw@cernet.edu.cn; dongenhuan@tsinghua.edu.cn; qli01@tsinghua.edu.cn).

Yuxi Zhu, Yixiang Zhang, Hanyi Peng, and Yixiao Wang are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: zyx23@mailstsinghua.edu.cn; zhangyix19@mailstsinghua.edu.cn; peng-hy21@mailstsinghua.edu.cn; yixiao-w20@mailstsinghua.edu.cn).

Kun Sun is with the Department of Information Sciences and Technology, George Mason University, Fairfax, VA 22030 USA (e-mail: ksun3@gmu.edu).

Menghao Zhang is with the School of Software, Beihang University, Beijing 100191, China (e-mail: zhangmenghao0503@gmail.com).

Jiang Li is with the Zhongguancun Laboratory, Beijing 100094, China (e-mail: lij@mail.zgclab.edu.cn).

Digital Object Identifier 10.1109/TIFS.2024.3442530

control. It is infeasible in many circumstances, particularly when there are numerous remote endpoints on the Internet that clients may communicate with. Second, existing approaches are mostly tightly coupled to specific network applications, such as browsers [40]. They fail to support obfuscating the TCP traffic for various applications. In practice, users may want the freedom of choosing which applications' TCP flows to be obfuscated for privacy and performance considerations. It is also desirable for users to set different privacy demands for TCP flows of different network applications. However, none of the existing traffic obfuscation solutions allow users to choose which applications to protect with what protection levels.

In this paper, we aim to explore a client-side and application-transparent approach to obfuscate encrypted TCP traffic for various applications. However, realizing such goals poses two significant challenges. First, we cannot add chaff packets or pad packets at will like existing approaches [29], [43] to obfuscate the uplink traffic. As we do not have the explicit cooperation of the remote side, the original contents of the communications cannot be correctly decoded. Also, we do not have the capability to directly control or modify the packets at the remote side to obfuscate the downlink traffic. Second, since different applications have different implementations and communication behaviors, it is challenging to transparently enforce TCP traffic obfuscation for them without affecting their functionalities and communication semantics. We may manually analyze and change the source code for each application to add traffic obfuscation operations; however, it is time-consuming and mostly impossible to access the source code of various applications. While modifying and recompiling the kernel to enforce traffic obfuscation seems like a general solution to support various applications, it is too complex and error-prone for users to adopt such a solution.

We show in this paper that it is possible to tackle the above challenges by presenting *Cactus*. It can reshape bidirectional TCP traffic at client side to achieve effective obfuscation while preserving the underlying communication semantics. The key insight is to leverage the inherent interaction and collaboration mechanisms of TCP protocol between two endpoints to introduce chaff packets, split packets, and limit the maximum size of packets. Cactus provides four fundamental operations to effectively obfuscate the uplink and downlink TCP traffic without affecting the communication semantics. For instance, Cactus can craft a chaff packet on the uplink and ensure it will not match the receiving window of the server. Consequently, the chaff packet is naturally disregarded by the server, without interfering with the normal communication process.

Additionally, it is crucial for Cactus to operate in an application-transparent manner, especially for various applications that require traffic obfuscation. Fortunately, the emerging eBPF technique offers a way for users to manipulate packets from different applications at the eBPF-TC kernel hook. By leveraging this feature, Cactus can transparently enforce traffic obfuscation across different applications without the need for kernel recompilation or modifications to the applications themselves. Specifically, Cactus injects customized traffic obfuscation eBPF programs at the eBPF-TC kernel hook, manipulating application packets in a way that triggers inherent TCP interaction and collaboration mechanisms. This approach effectively reshapes various application TCP flows. Moreover, to enforce different obfuscation levels for applications, Cactus automatically traces TCP flows for each application by leveraging the eBPF capability to attach a probe function to TCP socket creation functions. Whenever an application initiates a TCP flow, Cactus can precisely identify the application that generates the flow with the information collected from socket creation functions. Thus, Cactus can support users to choose which applications to conduct obfuscation operations and set suitable obfuscation levels for different applications.

We conduct extensive experiments to evaluate the effectiveness of Cactus on defeating typical encrypted TCP traffic analysis attacks in website fingerprinting and application identification. The results show that Cactus can effectively prevent DL-based traffic analysis attacks from inferring sensitive information over encrypted TCP flows. For instance, Cactus effectively reduces the accuracy of website fingerprinting from 86.25% to 25.66%. Similarly, the accuracy of application identification experiences a significant drop of 38.15% with Cactus. Furthermore, Cactus introduces low overheads. For instance, it only consumes additional 10% bandwidth to achieve 44.43% accuracy drop for website fingerprinting. It is also worth mentioning that Cactus introduces negligible additional latency for packet transmission.

In summary, we make the following contributions:

- We design Cactus, a client-side plug-in that protects the sensitive information of encrypted TCP traffic from being inferred by DL-based traffic analysis attacks.
- We leverage the inherent mechanisms of TCP protocol to design four fundamental operations that can effectively obfuscate bidirectional TCP traffic at client side while preserving the underlying communication semantics.
- We leverage the emerging eBPF technique to transparently enforce traffic obfuscation, allowing users to choose the applications conducting traffic obfuscation and customize the obfuscation level for each application.
- We conduct extensive experiments across various attack scenarios, including website fingerprinting and application identification, to verify the effectiveness of Cactus.

The rest of the paper is organized as follows. Section II describes the threat model, and presents the high-level design of Cactus. Section III provides the four fundamental TCP traffic obfuscation operations of Cactus. Section IV presents how to enforce the TCP traffic obfuscation with eBPF. Section V evaluates the effectiveness of Cactus. Section VI reviews related work. Section VII discusses the potential extensions and limitations of Cactus. Section VIII concludes the paper.

II. Threat Model and Cactus Design

In this section, we present the threat model for traffic analysis attacks. We then introduce the design goal and the design overview of Cactus.

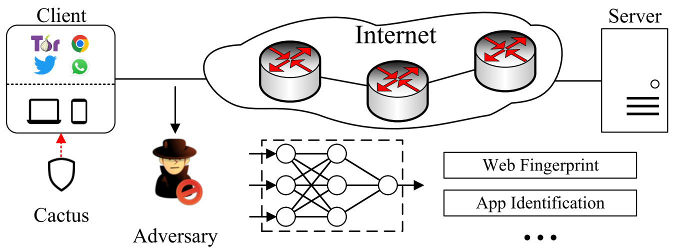


Fig. 1. Threat model.

A. Threat Model

We focus on the scenario where a passive adversary engages in traffic analysis attacks on a network device [48], [53], [54], as shown in Figure 1. The adversary intercepts encrypted TCP traffic and accesses encrypted packets exchanged between the client and server. We do not assume the adversary has the ability to decrypt these packets, but it can leverage the advanced DL-based traffic analysis methods [37], [48], [53], [54] to automatically extract the implicit and informative features from the packet size, direction, and timing information. These features can be leveraged to infer various information, such as the visited website [53], [54], applications that users use [5], [60], device identities [1], [2], operating systems [30], [57], and even the contents of VOIP calls [4], [16].

Considering that traffic analysis attacks can exploit information from both the uplink TCP traffic and the downlink TCP traffic between a client and a server, it is crucial for our defense to provide obfuscation for the bidirectional traffic. We do not require the capability to control remote networks or servers, or deploy network devices in local and remote networks. Neither do we require the capability to change source code of applications or recompile OS kernels. Our defense aims to obfuscate bidirectional encrypted TCP traffic for various applications in a client-side and application-transparent manner, which renders DL-based traffic analysis attacks ineffective.

B. Design Goal

Our defense aims to provide effective obfuscation for bidirectional TCP traffic without requiring any modification on the remote side. Meanwhile, our approach provides a unified traffic obfuscation manner for various applications without changing applications or recompiling OS kernels. It can provide users with the flexibility to customize the protected application and obfuscation level according to their specific requirements. In summary, we have two design goals:

1) Goal-1 (G1): Client-side Deployment: Despite the prevalence of two-side deployment in existing approaches [61], [62], its practical usage meets great challenges. Notably, the client's capability to control remote networks or servers is limited. It may be infeasible in many circumstances, especially when a client interacts with numerous remote endpoints scattered across the Internet. As a consequence, client-side deployment would be a better solution in practice. We aim to obfuscate bidirectional traffic while focusing solely on client-side deployment without affecting the underlying communication semantics.

2) Goal-2 (G2): Application-transparent Obfuscation: Existing approaches typically exhibit a strong dependency on specific applications [40], [43], primarily focusing on browsers. They lack the capacity to obfuscate TCP traffic originating from various applications transparently. Neither they support distinct traffic obfuscation levels for different applications that have diverse privacy protection requirements. We aim to provide an application-transparent obfuscation method, which empowers users to specify which application to enforce obfuscation operations and set different obfuscation levels for different applications.

C. Cactus Overview

To achieve the above design goals, we propose Cactus, a client-side and application-transparent plug-in aimed at obfuscating bidirectional encrypted TCP traffic for applications. As Figure 2 shows, Cactus consists of two main modules: user-defined obfuscation strategy module and TCP traffic obfuscation execution module. The former module enables the user to specify the obfuscation level for a given application. Then, it automatically tracks the flows associated with the given application and calculates the execution probabilities. The latter module leverages four fundamental obfuscation operations to reshape the bidirectional traffic. It executes the operations according to the execution probabilities delivered by the former module.

In order to achieve **G1**, Cactus introduces four fundamental operations to obfuscate the communication between the client and server. Two of these operations leverage the byte stream transmission of the TCP protocol to directly manipulate the uplink traffic without introducing additional delays, while the other two operations indirectly manipulate the downlink traffic by utilizing the flow control and retransmission mechanisms of the TCP protocol. For instance, Cactus can obfuscate the uplink traffic by dividing a packet into multiple smaller packets without causing any disruption to the server's receiving process. Besides, through truncating a received packet, Cactus can achieve indirect manipulation of the server, introducing the transmission of a new downlink packet. Section III details the design of the four fundamental operations.

In order to achieve **G2**, Cactus enables users to flexibly set different obfuscation levels for different applications, and enforces traffic obfuscation for application flows by leveraging the eBPF techniques. When given the root privilege of the operating system, the eBPF techniques provide Cactus with the ability to manipulate the application packets at the eBPF-TC (Traffic Control) hooks and track the information about the executed functions. Cactus tracks the flows for applications by attaching a probe function to the TCP socket creation function. Whenever an application initiates a TCP flow by invoking the socket creation function, the probe function will be automatically activated to obtain the process ID of the application and the flow information. Next, Cactus calculates distinct execution probabilities in accordance with obfuscation levels for different applications. Eventually,

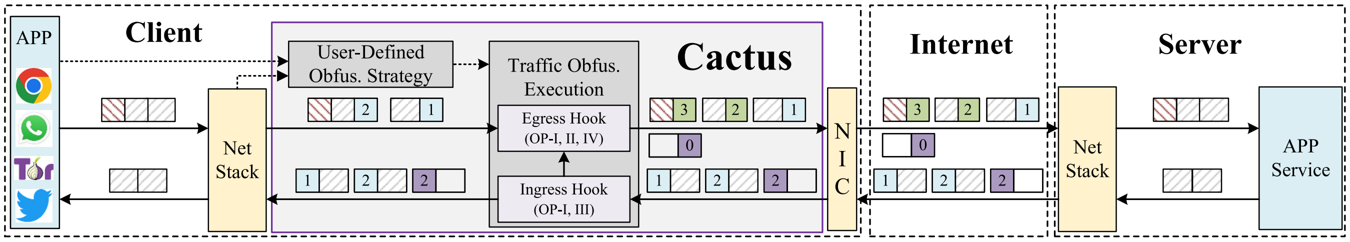


Fig. 2. The architecture of Cactus plug-in. Packets with blue headers represent the original application packets. The packets influenced by Cactus include various types, distinguished by their headers. Uplink packets with purple headers are identified as chaff packets, those with green headers are fragmented packets, and downlink packets with purple headers indicate retransmitted packets. The shading within the packet payload area highlights the actual data being transmitted.

it executes obfuscation operations for application flows at the eBPF-TC hooks based on execution probabilities. In our design, Cactus functions as a global module operating on the system and intercepting traffic from all processes at the eBPF-TC hooks simultaneously. Section IV details the design.

III. TCP Traffic Obfuscation Operation Design

This section provides the design of Cactus's four fundamental traffic obfuscation operations at client side. All these operations can effectively obfuscate encrypted TCP traffic without affecting the communication semantics of applications.

A. Obfuscation Operations for Uplink Traffic

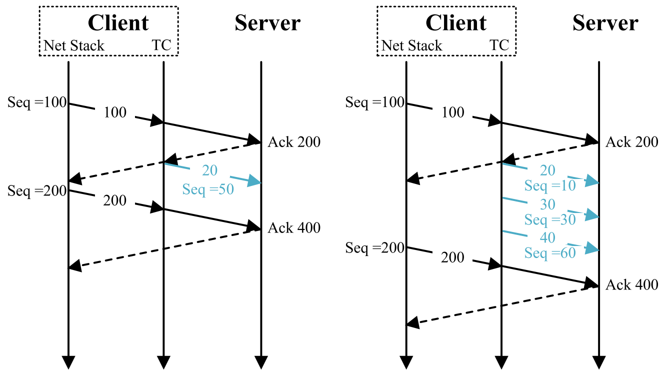
Building upon the byte stream delivery characteristics of the TCP protocol, our approach incorporates two fundamental operations aimed at reshaping uplink traffic to the server: chaff packet generation (**OP-I**) and packet fragmentation (**OP-II**). They are meticulously designed to ensure efficiency and the preservation of communication semantics.

To ensure optimal transmission efficiency, it is crucial that the operations involved in Cactus introduce smaller delays. We should avoid adopting operations that directly increase the delays of packets. If we directly shuffle the outgoing packets to introduce deliberate delays for traffic obfuscation, it adversely impacts the user experience of the application, rendering it undesirable.

In addition to optimizing efficiency, it is important for the operations to preserve communication semantics while adhering to client deployment restrictions. In particular, the enforcement of operations should not interfere with the normal function of applications. If we send a dummy packet with fabricated data to obfuscate traffic, it may corrupt application semantics and potentially cause the server to crash upon successful reception of the packet due to the unexpected information in the dummy packet.

1) Chaff Packet Generation: According to the basic mechanisms of TCP, a receiver keeps a record of received data at the receiver window and disregards any data that have already been received. This behavior inherent in TCP creates an opportunity for Cactus to devise Operation I (OP-I) that generates chaff packets without affecting the communication semantics.

OP-I involves creating a chaff packet with fake data and disguising the chaff packet as a previously received



(a) Single chaff packet. (b) Multiple chaff packets.

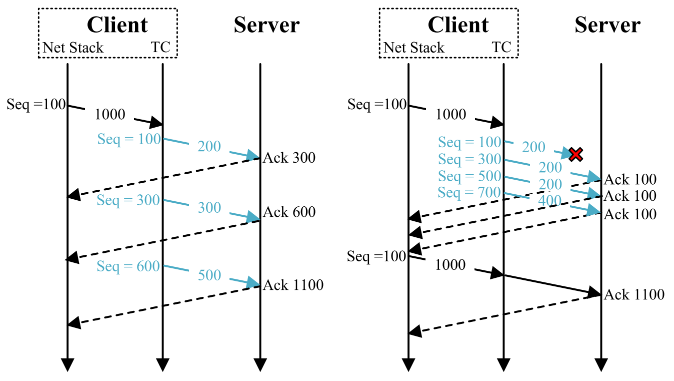
Fig. 3. Diagrams of OP-I.

packet by assigning it a proper sequence number. Specifically, the acknowledgment packet from the server contains an *acknowledgment number* which indicates the maximum sequence number of received data. Each time the client receives an acknowledgment packet, the client can craft a chaff packet and assigns it a sequence number of a value lower than the acknowledged number. Consequently, the chaff packet will be disregarded by the server since the assigned sequence number makes it seem like a previously received packet.

Figure 3 shows how the operation is designed. Figure 3a shows the diagrams of generating a single chaff packet. As the client receives a corresponding acknowledgment packet with an acknowledgment number of 200, it indicates that the server will only accept a packet with a sequence number of more than 200 in subsequent communications. Thus, Cactus generates a chaff packet to carry fake data of 20 bytes with a sequence number of 50. If no packet is lost, the server finally receives three packets with packet length sequences of [100, 20, 200]. However, as the sequence number of the chaff packet is smaller than 200, the chaff packet is disregarded by the server TCP/IP stack. Therefore, the fake data in the chaff packet will not be received by the applications running on the server. By leveraging this operation, we can flexibly introduce chaff packets in the TCP flows without affecting communication semantics for different applications. Furthermore, we can generate multiple chaff packets when the client receives an acknowledgment packet to achieve aggressive traffic obfuscation, which is shown in Figure 3b.

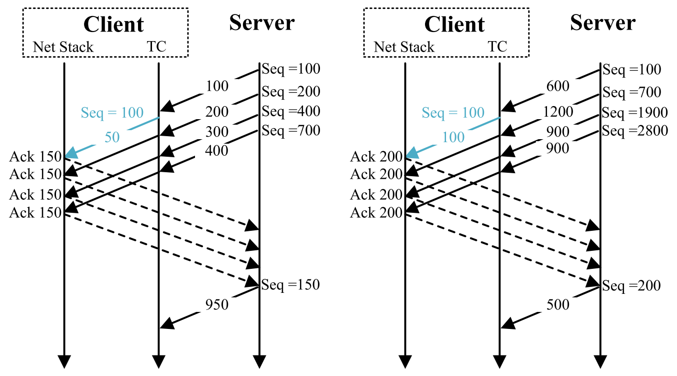
TABLE I
FUNDAMENTAL OPERATIONS OF CACTUS

Target Direction	Operation	eBPF Hook Point	Packet Size	Packet Direction	Timing
Uplink	Chaff Packet Generation (I)	TC-ingress & TC-egress	✓	✓	✓
	Packet Fragmentation (II)	TC-egress	✓	✓	✓
Downlink	Partial Data Uploading (III)	TC-ingress	✓	✓	✓
	Window Size Modification (IV)	TC-egress	✓	✓	✓



(a) Normal (b) Packet loss

Fig. 4. Diagrams of OP-II.



(a) Data in `swnd` is no more than MSS. (b) Data in `swnd` is more than MSS.

Fig. 5. Diagrams of OP-III.

OP-I introduces chaff packets into the uplink traffic with different size. Moreover, the increased number of uplink packets changes the packet direction information, and the additional intervals between the chaff packets and the original packets alter the packet timing information. Here, the term *interval* refers to the time period between the transmission of two consecutive network packets.

2) Packet Fragmentation: A TCP connection allows the utilization of multiple smaller packets to transmit the data originally contained in a single packet, while preserving the integrity of the communication semantics. This is made feasible by the ability of the receiving side to reconstruct the original packet from the smaller packets with the byte stream delivery mechanism inherent to TCP.

Cactus leverages the above mechanism to develop Operation II (OP-II) to fragment TCP packets, i.e. distributing the payload contained within a packet across multiple packets. OP-II employs a procedure that involves extracting the payload from a packet and organizing them into distinct subsets. These subsets are subsequently encapsulated into multiple smaller packets and transmitted individually. The TCP/IP stack of the server will utilize sequence numbers to ensure that the data are received and reassembled correctly. Consequently, applications receive the complete data as it was originally transmitted, despite the fragmentation that occurred during transmission.

Figure 4 shows that OP-II fragments a packet during the transmission. For simplicity, we consider that a packet with a payload length of 1000 is sent from the network stack of the client. Figure 4a displays the enforcement of OP-II in a network without packet loss. It distributes the packet's data into three packets with payload lengths of 200, 300, and 500, respectively. They are then transmitted to the server.

Figure 4b demonstrates the resilience of OP-II when facing packet loss. In the scenario where Cactus distributes the packet's data into four packets with payload lengths of 200, 200, 200, and 400, respectively, the first packet is lost in transit and the server receives the remaining three packets. Subsequently, the server informs the client of the lost data by sending duplicate acknowledgment packets. Due to the fast retransmission mechanism, the client will resend the lost packet. Consequently, the server ultimately receives four packets with a payload sequence of [200, 200, 400, 1000]. Although the traffic pattern is changed by Cactus, all data are reliably received by the server. By leveraging the operation, we can flexibly fragment packets in the TCP flows while preserving the communication semantics.

As OP-II distributes the data in a packet into several smaller packets, the packet size and direction information are directly affected by the increased packet number and smaller packet size. Additionally, the time intervals between the smaller packets serve to obfuscate the timing information of the traffic, further changing the traffic pattern.

B. Obfuscation Operations for Downlink Traffic

As Cactus enforces client-side deployment, we fail to directly control the downlink traffic at the server side. Fortunately, we can indirectly manipulate the downlink traffic by altering the state of the TCP connection at the client side. We present two fundamental operations to affect the downlink traffic based on the byte stream delivery and retransmission mechanisms of TCP protocol. These operations are partial data uploading (**OP-III**) and window size modification (**OP-IV**).

1) Partial Data Uploading: In a TCP connection, the network stack of the server maintains a Sender Window `swnd` to buffer unacknowledged data. According to the TCP specification [3], if the server receives multiple duplicate acknowledgment packets from the client, it typically indicates that specific unacknowledged data is lost. Consequently, the server will generate packets to retransmit the lost data.

Leveraging this characteristic of the TCP protocol, we provide Operation III (OP-III) to indirectly manipulate the server to retransmit data buffered in `swnd` by actively triggering acknowledgment packets at the client side. Specifically, OP-III truncates the received packet and uploads partial data to the network stack at the client side. Meanwhile, OP-III crafts corresponding duplicate acknowledgment packets to inform the server, simulating that partial data are lost. Consequently, the server will generate new packets to retransmit the "lost" data. OP-III thus indirectly introduces new packets to obfuscate the downlink traffic. In practice, the impact of OP-III can be changed by the size of data buffered in `swnd`. Assuming that the server receives duplicate acknowledgment packets from the client, following the typical TCP implementation [36], the server attempts to encapsulate all the unacknowledged data into a packet. However, in cases where the total size of data buffered in `swnd` exceeds the Maximum Segment Size (MSS), the server will send multiple packets to transmit the unacknowledged data. This process ensures that the length of each packet is less than or equal to MSS.

Figure 5a demonstrates the design of OP-III for packets with data stored in the `swnd` that do not exceed the MSS of 1460 bytes. For simplicity, we assume that the server sends four packets to the client with payload length sequences of [100, 200, 300, 400]. Meanwhile, the payload data are stored in the `swnd` of the server network stack. OP-III truncates the first packet and uploads only 50 bytes of data to the network stack at the client side. Subsequently, the client receives the next three packets. Meanwhile, OP-III generates three duplicate acknowledgment packets, which signals to the server that it only accepts the first 50 bytes of data and requests the server to retransmit the unacknowledged data. In response, the server transmits a packet containing all the unacknowledged data, totaling 950 bytes (50 + 200 + 300 + 400), back to the client. Five packets with payload lengths of 100, 200, 300, 400, and 950 are captured on the Internet. Consequently, the downlink traffic is obfuscated by introducing new packets with partial data uploading.

Figure 5b illustrates the design of OP-III for packets containing data stored in the `swnd` that exceed the MSS of 1460 bytes. We consider a scenario where the server sends four packets to the client, with payload length sequences of [600, 1200, 900, 900]. Meanwhile, the payload data are stored in the `swnd` of the server network stack. OP-III truncates the first packet and uploads only the first 100 bytes of data to the network stack at the client side. Subsequently, the client receives the next three packets. Meanwhile, OP-III generates three duplicate acknowledgment packets, which signals to the server that it only accepts the first 100 bytes of data and requests the server to retransmit the unacknowledged data. In response, the server is expected to retransmit all the unacknowledged data, totaling 3500 bytes (500 + 1200 + 900 + 900). However, the total length of the unacknowledged data exceeds the MSS of 1460 bytes. Following the typical TCP implementation [36], the server does not merge the data into a single packet. Instead, it will transmit four separate packets to accommodate the data. The first packet carries 500 bytes of data, while the remaining packets carry 1200, 900, and 900 bytes of data, respectively. Consequently, eight packets can be captured on the Internet, with payload length sequences of [600, 1200, 900, 900, 500, 1200, 900, 900].

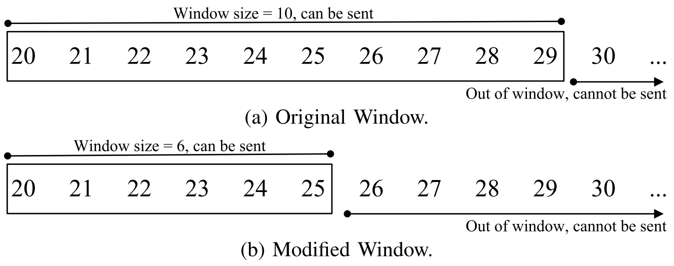

Window size = 10, can be sent
20 21 22 23 24 25 26 27 28 29 30 ...
Out of window, cannot be sent
(a) Original Window.
Window size = 6, can be sent
20 21 22 23 24 25 26 27 28 29 30 ...
Out of window, cannot be sent
(b) Modified Window.

Fig. 6. An example of how OP-IV affects the downlink packet by modifying the window size in the uplink packet.

OP-III utilizes the byte stream delivery and the retransmission mechanisms of TCP to indirectly manipulate the server's behavior on sending data. Instead of sending all data in a single packet, OP-III triggers the server to retransmit only a subset of data within a packet of arbitrary length. As a result, both the packet size and direction information are altered due to the presence of the crafted retransmitted packets. Furthermore, the introduction of the retransmitted packets also affects the timing information of packets. The unexpected inclusion of retransmission packets modifies the intervals between the original packets, reshaping the packet timing characteristics.

2) Window Size Modification: The TCP header contains a window size field that regulates the amount of data that can be transmitted before receiving acknowledgment packets. If the client sends a TCP packet to the server, the window size in the packet header will limit the maximum length of the subsequent downlink packets. Leveraging this TCP mechanism, we present Operation IV (OP-IV) that modifies the window size field in uplink packets to manipulate the length of downlink packets.

Figure 6 illustrates how OP-IV indirectly crafts the packet length of downlink traffic by manipulating the window size in the uplink packet. As shown in Figure 6a, if the server receives an uplink packet with a window size of 10, the server can generate a downlink packet carrying 10 bytes of data at most. However, if OP-IV modifies the window size of the uplink packet to 6 at the client side, the server can only generate a downlink packet carrying 6 bytes of data at most, as shown in Figure 6b.

OP-IV thus can obfuscate the downlink traffic at the client side. For instance, if the server intends to send data of 450 bytes, the data will typically be encapsulated into a single packet when receiving an uplink packet indicating a window size greater than 450. However, OP-IV can specify a window size of 250 in the uplink packet. Therefore, the server will be compelled to transmit at least two packets to accommodate these data. In this case, the payload of the first packet will be limited to 250 bytes. As a result, the packet size and direction information will be altered. Meanwhile, the additional packet interval of the two packets alters the packet timing information.

IV. TCP Traffic Obfuscation Enforcement

In this section, we initially present how users can enforce different obfuscation strategies for various applications. Next, we introduce the implementation of our traffic obfuscation operations using eBPF in an application-transparent manner.

A. User-Defined Traffic Obfuscation Strategy

Cactus enables users to configure different obfuscation levels for various applications for privacy and performance considerations. To achieve this capability, it initially tracks TCP flows specific to an application and then allows users to specify the desired obfuscation level with a demand parameter.

1) Tracking TCP Flows of Applications: As the obfuscation levels are configured for applications while the operations are executed for TCP flows, Cactus initially acquires the correlation between applications and flows. Cactus leverages the eBPF technique to automatically track the flows of any given application. It attaches a probe function to the TCP socket creation function and monitors which application establishes the TCP flows.

Specifically, Cactus attaches a probe function to the kernel function `inet_csk_accept`. Each time an application establishes a new TCP flow, `inet_csk_accept` will be invoked. Correspondingly, the probe function will be automatically activated to retrieve the application's process identification (PID) and the 5-tuple information of the new flow. Here, the 5-tuple information contains source and destination IP addresses, source and destination ports, and the transport layer protocol identifier. Based on the mapping between the application's PID and the 5-tuple information of the flows, Cactus can efficiently identify and track the application flows by inspecting packet headers.

2) Specifying the Traffic Obfuscation Level: Cactus allows users to modify a key demand parameter denoted by α for a given application to specify the obfuscation level. Here, α is a user-defined parameter that ranges from 0 to 1. If a user assigns a high value to α for a given application, Cactus will interpret this as an indication that the application has a correspondingly high obfuscation demand. By setting different α values for different applications, Cactus can enforce different obfuscation levels for TCP flows of an application. In addition to user demands, it is important to consider the network environment when executing the operations. For example, in a congested network, generating excessive chaff packets may severely exacerbate network congestion and lead to significant performance degradation.

Hence, we should explore an execution probability equation to control the execution of operations with the constraints of user demands and the network environment. The execution probability equation should meet two important requirements: 1) it enables Cactus to execute obfuscation operations more

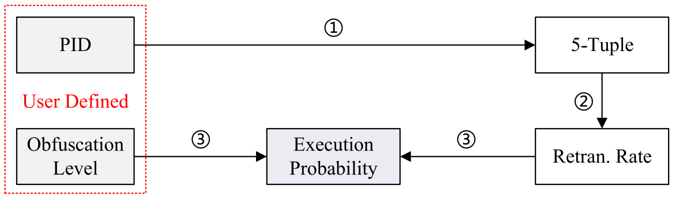


Fig. 7. The procedure for execution probability calculation.

frequently with a higher value of α. 2) if the retransmission rate is high, the operations should be executed less frequently to avoid significant performance degradation. Consequently, we design an equation for execution probability as follows:[1]

$$F_\alpha(r_{rts}) = \alpha(1 - r_{rts}). \tag{1}$$

According to the above equation, a higher α value can typically increase the execution probability, leading to more frequent execution of obfuscation operations. This increased frequency of operations enhances the variability of traffic patterns, resulting in a higher level of obfuscation for network flows.

Figure 7 shows the procedure on how we calculate the execution probability. Considering the scenario that a user defines an obfuscation level α for an application. To calculate execution probabilities for the application flows, we follow the three steps: (1) collecting the 5-tuple information of flows associated with the application by attaching a probe function; (2) retrieving the retransmission rates of these flows from the TCP sockets associated with the flows; (3) calculating the execution probabilities for flows according to Equation 1. Additionally, considering the dynamic nature of the network environment, we periodically gather the retransmission rates of the flows to update the execution probabilities. In our implementation, we store the execution probability in eBPF maps so that traffic obfuscation programs at the eBPF-TC hooks can quickly retrieve the probability information.

Algorithm 1 details how obfuscation is enforced with a given obfuscation level α for an application. For a host deployed with Cactus $host$, the given obfuscation level for the application α, and the maximum segment size MSS, obfuscation is enforced when an application packet is injected into eBPF-TC. The procedure commences by querying the host for the retransmission rate of the flow (r_{rts}) associated with the packet. Subsequently, it computes the execution probability (exe_p) based on α and r_{rts}. Lines 3 to 18 address the scenario where pkt should be received by $host$. In Lines 4 to 12, when pkt is an acknowledgment packet, OP-I will be executed with a probability of exe_p. OP-I begins by generating a chaff packet ch_pkt by cloning pkt. The algorithm then randomizes the size of ch_pkt through padding and truncation. Next, it sets the sequence number of ch_pkt to the acknowledgment number of pkt minus MSS. It then swaps the source and destination information in the TCP header. Finally, $host$ sends ch_pkt as a chaff packet to obfuscate uplink traffic. In Lines 13 to 16, when pkt has a payload size greater than 1, OP-III will be executed to randomly truncate pkt with a probability of exe_p. Lines 18 to 30 address the scenario where the

[1]Other equations can also be available if meeting the above requirements.

Algorithm 1 Obfuscation Enforcement for an Application

```
Input: $pkt$ : an application packet;
       $\alpha$ : Obfuscation level for the application;
       $host$ : a host deployed with Cactus;
       $MSS$ : Maximum Segment Size
1: $r_{rts} \leftarrow host.query(pkt.header)$    ▷ Retransmission rate
2: $exe\_p \leftarrow \alpha * (1 - r_{rts})$
3: if $pkt.dip = host.ip$ then ▷ $pkt$ is received by the host.
4:     if $Random(0,1) < exe\_p$ and $pkt.flag.ack = true$
           then    ▷ Necessary condition of OP-I
5:         $ch\_pkt \leftarrow pkt.copy()$
6:         $ch\_pkt.seq\_num \leftarrow pkt.ack\_num - MSS$
7:         $ch\_pkt.payload.padding(MSS)$
8:         $ch\_pkt.payload.truncate(RandomInt(0, MSS))$
9:         $exchange(ch\_pkt.sip, ch\_pkt.dip)$
10:        $exchange(ch\_pkt.sport, ch\_pkt.dport)$
11:        $host.send(ch\_pkt)$
12:    end if
13:    if $Random(0,1) < exe\_p$ and $pkt.payload.len > 1$
           then    ▷ Necessary condition of OP-III
14:        $L \leftarrow RandomInt(0, pkt.payload.len)$
15:        $pkt.payload.truncate(0, L)$
16:    end if
17:    $host.receive(pkt)$
18: else    ▷ $pkt$ is sent from the host.
19:    if $Random(0,1) < exe\_p$ and $pkt.wnd\_size >$
           $MSS$ then    ▷ Necessary condition of OP-IV
20:        $pkt.wnd\_size \leftarrow RandomInt(0, MSS)$
21:    end if
22:    if $Random(0,1) < exe\_p$ and $pkt.payload.len > 1$
           then    ▷ Necessary condition of OP-II
23:        $adt\_pkt \leftarrow pkt.copy()$
24:        $L \leftarrow RandomInt(1, pkt.payload.len)$
25:        $pkt.payload.truncate(0, L)$
26:        $adt\_pkt.payload.truncate(L, pkt.payload.len)$
27:        $host.send(pkt)$
28:        $pkt \leftarrow adt\_pkt$
29:    end if
30:    $host.send(pkt)$
31: end if
```

packet is sent by the host. In Lines 18 to 21, when pkt has a window size wnd_size greater than MSS, OP-IV will be executed to modify its window size to a smaller value with a probability of exe_p. In Lines 22 to 29, if pkt has a payload size greater than 1, OP-II will be executed with a probability of exe_p. OP-II begins by generating an additional packet adt_pkt by cloning pkt. Then, it generates a random integer value L ranging from 1 to the payload size of pkt. Subsequently, it truncates both pkt and adt_pkt. Here, pkt retains the first L bytes of payload, while adt_pkt retains the last $(pkt.payload.len - L)$ bytes of payload. Finally, it sends pkt and assigns adt_pkt to pkt.

B. Traffic Obfuscation Implementation With eBPF

In order to effectively obfuscate the bidirectional TCP traffic, we implement the four operations at the eBPF-TC ingress and egress hooks.[2] The necessary execution conditions for these operations are shown in Table II. The operations are triggered when the client receives or sends specific types of packets. When the necessary execution condition for an operation is met, Cactus retrieves the corresponding execution probability from the eBPF maps. The operation is then executed with the retrieved execution probability. In the following, we detail the eBPF implementations of the four obfuscation operations.

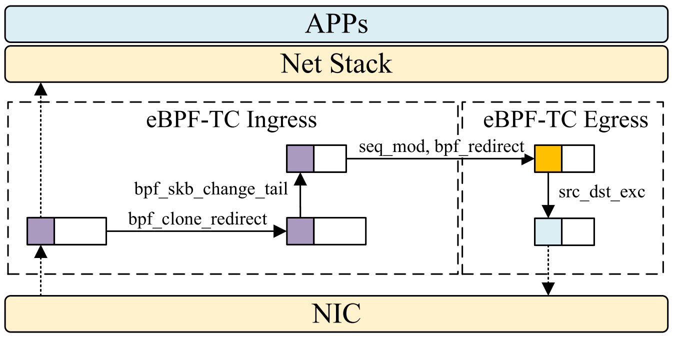


Fig. 8. The eBPF implementation of OP-I at the eBPF-TC ingress and egress hooks.

TABLE II
NECESSARY EXECUTION CONDITION OF OPERATIONS

Operation	eBPF Hook Point	Necessary Condition
OP-I	TC-ingress	Receiving acknowledgment packets
OP-II	TC-egress	Sending packets with data
OP-III	TC-ingress	Receiving packets with data
OP-IV	TC-egress	Sending packets with effective wnd_size

1) eBPF Implementation of OP-I: Figure 8 illustrates the procedure of generating a chaff packet at the eBPF-TC ingress and egress hooks. In the event that the client receives a packet containing a valid acknowledgment number, Cactus utilizes the `bpf_clone_redirect` helper to generate a replica of the packet. Subsequently, it invokes the `bpf_skb_change_tail` helper to alter the length of the replicated packet. The length of the packet payload is adjusted to a randomized value, which ranges from 0 up to the Maximum Segment Size (MSS) of the TCP connection. Following this modification, Cactus modifies its sequence number to ensure that the sum of this value and the payload length is smaller than the acknowledgment number present in the original packet. By using the `bpf_redirect` helper, Cactus redirects the modified packet to the eBPF-TC egress hook point. Furthermore, it exchanges the source and destination information in the TCP header of the modified packet. Finally, the modified packet is transmitted to the server as a chaff packet. By repeating the process, Cactus can generate multiple chaff packets.

2) eBPF Implementation of OP-II: Figure 9 depicts the process of fragmenting a packet into two smaller packets at the eBPF-TC egress hook. Assuming a packet is transmitted from the client, its payload can be considered to consist of two subsets. Cactus employs the `bpf_clone_redirect` helper function to create a replicated packet as the initial step.

[2]For more detailed information about the eBPF technique, refer to the eBPF tutorial available at https://docs.cilium.io/en/latest/bpf/

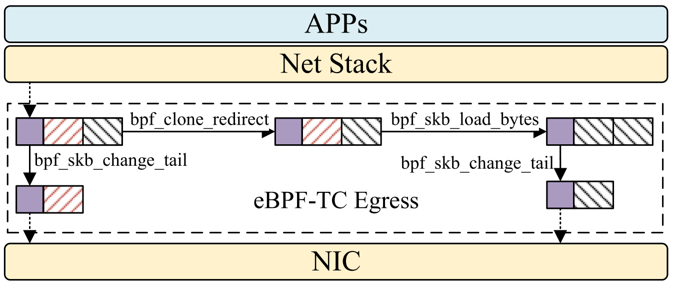


Fig. 9. The eBPF implementation of OP-II at the eBPF-TC egress hook.

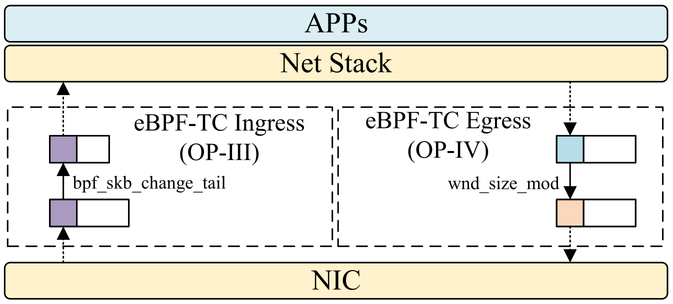


Fig. 10. The eBPF implementation of OP-III at the eBPF-TC ingress hook and OP-IV at the eBPF-TC egress hook.

Subsequently, it leverages the `bpf_skb_change_tail` helper function to truncate the original packet while preserving the front subset within it. Here, the size of the front subset ranges from 1 to the original packet length. Additionally, it overwrites the payload of the replicated packet with the latter subset by invoking `bpf_skb_load_bytes` helper. Next, Cactus applies the same `bpf_skb_change_tail` helper function to truncate the replicated packet, ensuring that the latter subset is appropriately retained. Consequently, the server can receive the complete data in the original packet after receiving the two truncated packets. By repeating the process, a packet can be split into multiple segmented packets.

3) eBPF Implementation of OP-III: Figure 10 illustrates the implementation of OP-III at the eBPF-TC ingress hook. When the client receives a packet with payload at the eBPF-TC ingress hook, Cactus truncates the packet by invoking the `bpf_skb_change_tail` helper function. Subsequently, the modified packet is passed to the network stack for further processing. As a result of the intentional data truncating, the client triggers the retransmission mechanism, prompting the server to retransmit the dropped data.

4) eBPF Implementation of OP-IV: Figure 10 presents the implementation of OP-IV at the eBPF-TC egress hook. In the event that the client transmits a packet that includes a valid window size, OP-IV modifies the window size to a smaller value at the eBPF-TC egress hook. Subsequently, the packet is forwarded to the server, thereby limiting the maximum size of the received packets.

V. Evaluation

In this section, we evaluate the effectiveness of Cactus on defeating typical encrypted TCP traffic analysis attacks, including website fingerprinting and application identification.

A. Experiments on Website Fingerprinting

1) Web Traffic Collection: We leverage 20 sender hosts in China as the clients to visit popular websites with the Selenium web driver [45] driving the Firefox browser. We collect web traffic without deploying Cactus at the beginning. Then, we deploy Cactus to obfuscate the traffic generated by the Firefox browser. We collect 1,462,961 TCP flows by visiting ten popular websites, including Adobe, Baidu, Bilibili, Microsoft, QQ, Taobao, Warner Bros, WordPress, Zhihu, and Zoom. Hence, there are ten labels in this dataset.

2) Model Training and Testing: We consider four typical deep learning models in our experiments, i.e. Convolutional Neural Network (CNN) [31], Long-Short Term Memory (LSTM) [26], and Deep Fingerprinting (DF) [53], and Robust Fingerprinting (RF) [49]. The first two models achieve good results on website fingerprinting [48]. DF effectively improves the performance of website fingerprinting in the scenario where the traffic is mixed with randomly padding packets. RF can effectively fingerprint the website even under various obfuscation approaches. In the experiments, we first train and test them using the flows collected without deploying Cactus. This establishes a baseline for comparison. Subsequently, to demonstrate the effectiveness of Cactus in obfuscating website traffic, we then train and test these models using the flows gathered after deploying Cactus. This approach assumes that attackers may be aware of the deployment of Cactus when conducting traffic analysis. Here, the ratio of the number of flows in the training set, validation set, and testing set is 6:2:2 for experiments with different α. We extract packet size, direction, and interval as representative features. Furthermore, as flows in real networks are naturally unbalanced [35], [37], we use the original ratio of different flows to train and test the existing models on website traffic classification. We utilize an RTX A6000 GPU for the training and testing in all the experiments.

3) Experimental Results: We apply two metrics, i.e., accuracy (AC) and macro-F1-Score (F1) to present our experimental results. Here, we compare different obfuscation levels by evaluating the AC and F1 performance of traffic analysis attacks on obfuscated flows. The low performance typically indicates a high obfuscation level for the flows. Table III shows the effectiveness of Cactus on defeating website fingerprinting with different obfuscation parameter α. More than 82% accuracy and F1-Score can be achieved without deploying Cactus, i.e. $\alpha = 0$. Particularly, DF achieves 92.00% accuracy and 91.80% F1-Score. However, we observe that the accuracy and F1-Score of all models consistently drop when α increases. Despite a small α of 0.02 in Cactus, the average accuracy decreases from 86.25% to 66.74%. When the α is set as 0.1, the average accuracy decreases by 60.60% and the average F1-Score decreases by 63.05%. The experimental results demonstrate that Cactus can effectively obfuscate website traffic even with small α, and α indeed provides users with the ability to control the obfuscation level. With a high α, the obfuscated flows are difficult to classify due to the high obfuscation level. Furthermore, we conduct an extensive evaluation of the transmission overhead in Section V-D to

TABLE III
PERFORMANCE ON DEFEATING WEBSITE FINGERPRINTING. LOWER AC AND F1 REPRESENT A BETTER DEFENSE

Model	Metric	W/o Cactus	With Cactus				
		α = 0	α = 0.02	α = 0.04	α= 0.06	α = 0.08	α = 0.10
DF	AC	85.76%	77.29%(↓8.47%)	72.57%(↓13.19%)	51.56%(↓34.20%)	44.79%(↓40.97%)	36.63%(↓49.13%)
	F1	85.06%	75.40%(↓9.66%)	69.13%(↓15.93%)	44.93%(↓40.13%)	32.10%(↓52.96%)	30.69%(↓54.37%)
CNN	AC	84.70%	53.96%(↓30.74%)	53.37%(↓31.33%)	26.04%(↓58.66%)	28.47%(↓56.23%)	18.23%(↓66.47%)
	F1	84.07%	53.20%(↓30.87%)	51.58%(↓32.49%)	24.13%(↓59.94%)	24.34%(↓59.73%)	16.85%(↓67.22%)
LSTM	AC	82.54%	48.13%(↓34.41%)	41.19%(↓41.35%)	21.35%(↓61.19%)	25.00%(↓57.54%)	16.15%(↓66.39%)
	F1	82.50%	45.44%(↓37.06%)	40.12%(↓42.38%)	20.40%(↓62.10%)	19.30%(↓63.20%)	14.88%(↓67.62%)
RF	AC	92.00%	87.58%(↓4.42%)	74.80%(↓17.20%)	68.33%(↓23.67%)	50.50%(↓41.50%)	31.61%(↓60.39%)
	F1	91.80%	84.97%(↓6.83%)	72.76%(↓19.04%)	64.51%(↓27.29%)	46.54%(↓45.26%)	28.82%(↓62.98%)
On average	AC	86.25%	**66.74%(↓19.51%)**	**60.48%(↓25.77%)**	**41.82%(↓44.43%)**	**37.19%(↓49.06%)**	**25.66%(↓60.60%)**
	F1	85.86%	**64.75%(↓21.11%)**	**58.40%(↓27.46%)**	**38.49%(↓47.37%)**	**30.57%(↓55.29%)**	**22.81%(↓63.05%)**

understand the impact of Cactus on the transmission of website visits.

B. Experiments on Application Identification

1) Application Traffic Collection: We leverage 20 hosts to run popular applications and collect their traffic. Initially, we capture the traffic without deploying Cactus. Subsequently, we deploy Cactus to introduce obfuscation for the application traffic. We collect 221,933 flows by running five popular applications, including QQmusic, TencentDoc, YoudaoNote, and NeCmusic. Hence, there are four labels in this dataset.

2) Model Training and Testing: We follow the experimental setting in Section V-A to evaluate the effectiveness of Cactus on defeating application identification.

3) Experimental Results: Table IV shows the effectiveness of Cactus on defeating application identification with different obfuscation parameter α. Without deploying Cactus, the four models achieve more than 93% accuracy and F1-Score on average. However, a significant drop occurs when Cactus is enabled. For instance, the accuracy of CNN is dropped from 91.81% to 48.33% when α is set as 0.1. The experimental results demonstrate that Cactus is effective for obfuscating application traffic.

C. Analysis on Feature Distribution

The layer before the last output layer of a deep learning model will generate a feature vector that reflects the implicit features of the input traffic. We hence can explore the implicit features with feature vectors extracted by deep learning models to demonstrate that Cactus effectively obfuscates the traffic. We use website flows to show our results since experiments with the application flows present similar results. We extract the feature vectors of flows collected with different α to check whether Cactus could effectively obfuscate the traffic. To better illustrate how the feature vectors change with the increasing of α, we apply the t-SNE [59] method to project high-dimensional feature vectors extracted by the DF model into 2D vectors. For simplicity, we only take DF as an example for illustration since other models show similar results like DF. We draw the vectors in 2D space, which is shown in

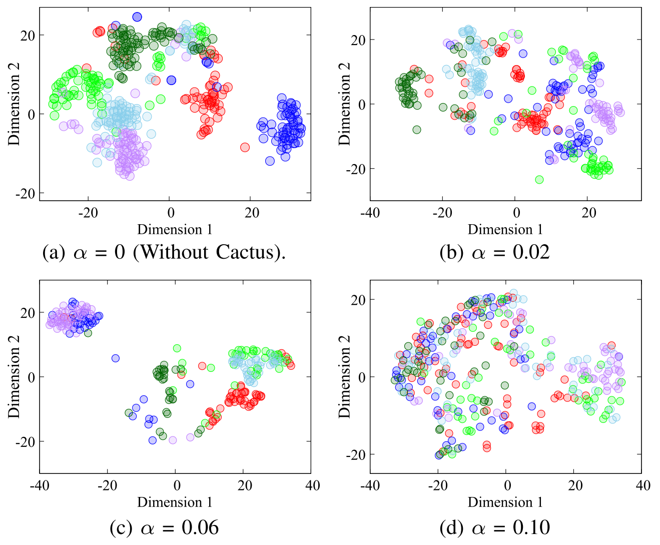


Fig. 11. Feature distribution of flows under different α.

Figure 11.[3] When Cactus is not enabled, i.e., $\alpha = 0$, there is a clear separation between feature vectors of different classes, while those belonging to the same class appear to be closely clustered. However, when we increase α, the boundary between classes becomes less distinct. This is evident in the observation that certain feature vectors, despite belonging to different classes, appear to be in close proximity, making it challenging to discern their true classes. This indicates that users can achieve a high obfuscation level for a flow by setting a high α.

D. Overhead

1) Bandwidth and Latency Overhead: We evaluate the bandwidth and latency overhead introduced by Cactus. We use the experiments with website flows to show the results. Note that experiments with other TCP flows present similar results. Figure 12a shows the additional bandwidth introduced by Cactus. With the increase of α, more bandwidth is consumed during the transmission. When α is set to 0.1,

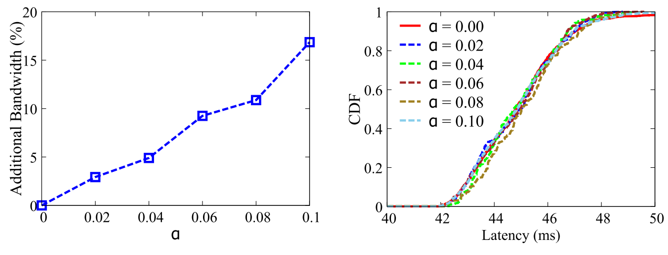
[3]Here, we show the vectors of six typical websites for clear presentation.

TABLE IV
PERFORMANCE ON DEFEATING APPLICATION IDENTIFICATION. LOWER AC AND F1 REPRESENT A BETTER DEFENSE

Model	Metric	W/o Cactus	With Cactus				
		α = 0	α = 0.02	α = 0.04	α= 0.06	α = 0.08	α = 0.10
DF	AC	92.08%	89.17%(↓2.91%)	87.64%(↓4.44%)	74.31%(↓17.77%)	70.93%(↓21.15%)	55.63%(↓36.45%)
	F1	92.03%	89.28%(↓2.75%)	87.45%(↓4.58%)	74.42%(↓17.61%)	71.29%(↓20.74%)	51.01%(↓41.02%)
CNN	AC	91.81%	84.59%(↓7.22%)	79.31%(↓12.50%)	64.17%(↓27.64%)	59.39%(↓32.42%)	48.33%(↓43.48%)
	F1	91.83%	84.42%(↓7.41%)	79.55%(↓12.28%)	64.21%(↓27.62%)	58.97%(↓32.86%)	46.34%(↓45.49%)
LSTM	AC	93.75%	83.38%(↓10.37%)	81.94%(↓11.81%)	68.33%(↓25.42%)	59.09%(↓34.66%)	61.67%(↓32.08%)
	F1	93.75%	83.63%(↓10.12%)	81.70%(↓12.05%)	68.50%(↓25.25%)	58.91%(↓34.84%)	60.71%(↓33.04%)
RF	AC	97.00%	95.77%(↓1.23%)	80.17%(↓16.83%)	68.00%(↓29.00%)	65.36%(↓31.64%)	56.41%(↓40.59%)
	F1	97.19%	95.10%(↓2.09%)	79.17%(↓18.02%)	67.88%(↓29.31%)	63.50%(↓33.69%)	57.08%(↓40.11%)
On average	AC	93.66%	**88.23%(↓5.43%)**	**82.27%(↓11.40%)**	**68.70%(↓24.96%)**	**63.69%(↓29.97%)**	**55.51%(↓38.15%)**
	F1	93.70%	**88.11%(↓5.59%)**	**81.97%(↓11.73%)**	**68.75%(↓24.95%)**	**63.17%(↓30.53%)**	**53.79%(↓39.92%)**

Cactus consumes about 17% additional bandwidth. It is because Cactus introduces chaff packets in the uplink flows and the retransmitted packets in the downlink flows for traffic obfuscation. However, such bandwidth consumption is acceptable in practice. Besides, Cactus allows users to flexibly set α to tradeoff the obfuscation level and the additional bandwidth consumption. Besides, we measure the latency from the time when the client sends a packet to the time when the client receives the acknowledgment packet. The latency distribution is depicted in Figure 12b. It is noteworthy that the latency remains relatively stable regardless of the increase of α in Cactus. This can be attributed to the fact that the four operations implemented in Cactus do not introduce significant delays for packets. Instead, variations in latency are likely due to minor fluctuations in the network environment.

2) *Total_Time and Accuracy Trade-off:* Cactus leverages various TCP mechanisms to effectively obfuscate bidirectional traffic at client side. These operations may affect the congestion control and flow control of TCP connections, leading to a potential increase in data transmission time. Hence, we use a popular tool named `curl` [18] to measure how Cactus affects the data transmission time with different obfuscation levels. Here, `curl` applies the metric *total_time* to indicate the time from the sending of the first byte to the receiving of the last response from the server, which can reflect the data transmission time. Following the settings in Section V-A, we set the α from 0 to 0.1 in Cactus and measure the total_time of visiting the Google website. For each α, we measure the total_time 100 times and record the average total_time. Figure 13 presents the trade-off between the average total_time and the average accuracy of DL models with Cactus for different obfuscation levels. We can see that the total_time increases as the obfuscation level α grows. Simultaneously, Cactus becomes more effective in countering traffic analysis attacks (i.e., more accuracy drops). We observe that when α is set to a value of 0.06, the total_time increases slightly from 359 ms to 438 ms, and the accuracy significantly decreases by 34%. Furthermore, even if a user sets α to 0.1 for high-level traffic obfuscation, our results show it only increases the total_time to 650 ms. According to Google's



(a) Additional Bandwidth. (b) Latency.

Fig. 12. Bandwidth and latency overhead for Cactus.

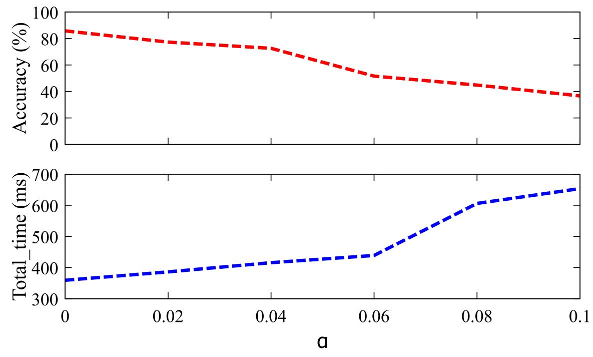


Fig. 13. Trade-off between total_time and accuracy.

report [24], 800 ms is an acceptable time for users. Note that the parameter α is essential for balancing data transmission performance and obfuscation levels, allowing users to tailor settings to their specific needs. For applications demanding real-time, fast data transmission, like voice communication, a smaller α is preferable. Conversely, for applications with less urgent needs for real-time data transmission, such as file downloads, a larger α can be employed to enhance privacy without notably affecting the user experience.

E. Ablation Study

We conduct ablation experiments by removing one of the four fundamental operations in Cactus each time. Table V shows the results of our ablation experiments. When we apply all the four operations in Cactus to obfuscate traffic, the accuracy decreases by 49.13%. However, the accuracy drop

TABLE V
ABLATION STUDY ON DIFFERENT OPERATIONS

OP-I	OP-II	OP-III	OP-IV	AC	F1
×	×	×	×	85.76%	85.06%
×	✓	✓	✓	72.69%(↓13.07%)	70.71%(↓14.35%)
✓	×	✓	✓	53.57%(↓32.19%)	47.90%(↓37.16%)
✓	✓	×	✓	46.09%(↓39.67%)	41.86%(↓43.20%)
✓	✓	✓	×	50.00%(↓35.76%)	39.48%(↓45.58%)
✓	✓	✓	✓	36.63%(↓49.13%)	30.69%(↓54.37%)

ranges from 13.07% to 39.67% when removing one of the four operations. We can see that any of the four operations is helpful to obfuscate traffic. Furthermore, we can see that the minimal drop in accuracy is 13.07% when removing OP-I. Thus, OP-I plays the most important role in traffic obfuscation.

VI. Related Work

A. Traffic Analysis Attacks

Traffic analysis attacks aim to infer sensitive information from network traffic by analyzing metadata such as packet timing, size, and direction. These attacks serve as the foundation for various tasks, including application identification [5], [58], [60], [66], website fingerprinting [21], [28], [48], [51], [53], [54], user profiling [17], [23], operating system identification [30], [57], and IoT device identification [2], [38]. Recent studies [17], [34], [39], [51], [53], [54], [64] focus on automatic feature extraction and employ Deep Learning (DL) to execute the attacks using raw traffic inputs.

Cao et al. [11] utilize DL models to fingerprint applications in software-defined networking by analyzing packet length sequences of TLS encrypted control traffic. Shen et al. [52] fingerprint decentralized applications by employing graph neural networks to analyze packet length sequences of encrypted flows. Rimmer et al. [48] automatically fingerprint websites by analyzing direction sequences of cells in encrypted Tor packets. Similarly, Sirinam et al. [53] propose Deep Fingerprint, which learns representations from packet direction sequences of encrypted Tor packets to achieve accurate website fingerprinting. Cherubin et al. [15] further evaluate website fingerprinting attacks in real-world network environments. In addition, traffic classification approaches [27], [35], [37] and malicious flow detection approaches [20], [42] are available for traffic analysis attacks.

Even though the above approaches achieve excellent results in inferring sensitive information from the encrypted TCP traffic, our experiments demonstrate that Cactus can effectively defend against typical DL-based traffic analysis attacks. Indeed, Xie et al. [63] present Rosetta to make DL models aware of regular changes with TCP semantics. Despite this advancement, Cactus is effective in defending against Rosetta as Appendix A presents.

B. Countermeasures for Traffic Analysis Attacks

Various approaches [10], [12], [33], [41], [44], [50] have been proposed to defeat traffic analysis attacks. They typically deploy proxies for specific applications or modify

TABLE VI
COMPARISON OF OBFUSCATION APPROACHES

Defense	Only Client-side Deployment	Support Various Applications	Target Traffic
QCSD [55]	✓	×	QUIC
HTTPOS [40]	✓	×	HTTP
FRONT [22]	×	×	Tor
GLUE [22]	×	×	Tor
WTF-PAD [29]	×	×	Tor
Walkie-talkie [61]	×	×	Tor
CS-BuFLO [8]	×	×	SSH, Tor
Traffic Morphing [62]	×	×	HTTP, VoIP
Loopix [44]	×	✓	Any
TARANET [13]	×	✓	Any
Ditto [41]	×	✓	Any
Cactus	✓	✓	TCP

the specific applications to enable various traffic obfuscation operations [8], [9], [21], [22], [29], [40], [43], [55], [61], [62], such as splitting and padding packets [62], generating dummy packets [29]. Nasr et al. [43] use generative adversarial networks to perturb traffic with minimal overhead. Several approaches [8], [9], [21] regularize the traffic features by fixing the packet size and interval. Besides, recent studies [6], [12], [13], [41], [44] rely on specific network devices, such as programmable switches, to enable efficient traffic obfuscation.

Although these approaches can effectively obfuscate TCP traffic patterns, they require two-side deployment to decode the obfuscated flows correctly. However, modifying the remote endpoints or deploying network devices on the remote networks is typically outside the client's control in practice. Several approaches [40], [55] have been provided to obfuscate traffic at the client side. Luo et al. [40] implement a browser proxy to obfuscate encrypted web traffic by modifying packet size, packet timing, web object size, and flow size at the client side. Smith et al. [55] overwrite the browser library to reshape QUIC traffic by padding packets, adding chaff packets, splitting, and delaying the packets at the client side. However, these approaches are tightly coupled to the web traffic and the specific network application, i.e. browsers. They fail to support obfuscating the TCP traffic for various applications.

We compare Cactus with the existing approaches in Table VI. Cactus can achieve bidirectional TCP traffic obfuscation at the client side. Meanwhile, it provides traffic obfuscation for various applications, and enables users to select the applications conducting traffic obfuscation and customize the obfuscation level for each application.

VII. Discussion

A. The Extension of Cactus for Non-TCP Encrypted Traffic

While initially designed for encrypted TCP traffic, Cactus can be extended to non-TCP protocols that have similar mechanisms like TCP. A typical protocol is QUIC, which incorporates a retransmission mechanism to ensure reliable communication. By leveraging the capabilities of eBPF programs to analyze QUIC packets, Cactus can identify a QUIC flow and intentionally drop its packets. This action can trigger the server to initiate the retransmission of the lost packets, introducing obfuscation into the traffic pattern. However, it should be noted that Cactus cannot be directly applied to QUIC or any non-TCP protocols due to the difference in design. The development of a similar approach for QUIC or any non-TCP protocol would indeed require a comprehensive understanding of the inner working mechanisms of the specific protocols. To effectively reshape the traffic of QUIC or other non-TCP protocols without disrupting their normal functionalities, it is essential to delve into the intricate details of each protocol's unique characteristics.

B. Deploying Cactus at Server Side

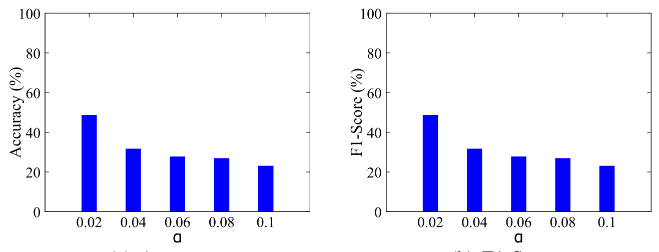
Even though Cactus is designed to be deployed at client side initially, it is feasible to deploy Cactus at the server side. In the scenario where an application service provider aims to prevent sensitive information from being inferred through its encrypted TCP traffic, it can deploy Cactus at the server side. Unlike existing approaches [22], [43], [61] that require upgrades for applications both on the server side and the client side, Cactus does not require any upgrades for the users' application software.

C. Considerations on Reconstructing Original Flows

An advanced adversary may attempt to reconstruct original flows by analyzing TCP packets to eliminate the traffic obfuscation that Cactus enforces. However, it is infeasible in practice. First, the original uplink flow is hard to be reconstructed since the packet fragmentation operation (OP-II) brings significant difficulties for the attacker. The attackers cannot know which packets are fragmented by analyzing the TCP packets, failing to reconstruct the original packet by possibly merging packets. Second, the reconstruction of the original downlink flow also poses a significant challenge due to the arbitrary reshaping capabilities of Cactus through OP-IV that manipulates the window size. By manipulating the window size, Cactus can alter the traffic pattern at will, making it difficult for the attacker to accurately reconstruct the original downlink flows.

D. Considerations on Obfuscation in Low-Volume Packet Exchanges

The effectiveness of adding chaff and fragmented packets is affected by the total number of packets transmitted between the client and server. In scenarios where only a small number of packets are exchanged, the limited number of chaff and fragmented packets may not sufficiently obfuscate key information, potentially exposing it to adversaries. To mitigate this, the frequency of these operations can be increased. Two strategies are available: one approach is to set a higher α value for the flows, which increases the likelihood of these operations being executed. Alternatively, multiple chaff and fragmented packets can be generated by repeatedly executing operations OP-I and OP-II each time a packet is received or sent by the client. This method would help increase the variability in traffic patterns, enhancing the overall obfuscation effectiveness.



(a) Accuracy. (b) F1-Score.

Fig. 14. The effectiveness of Cactus in defending against Rosetta.

VIII. Conclusion

We present Cactus, a client-side plug-in aimed at safeguarding encrypted TCP traffic against DL-based traffic analysis attacks. By leveraging the inherent interaction and collaboration mechanism of TCP, Cactus provides four fundamental operations to effectively obfuscate bidirectional traffic while preserving the communication semantics. Besides, with the emerging eBPF technique, Cactus empowers users to specify which applications to conduct traffic obfuscation and what obfuscation level for each application. To validate the effectiveness of Cactus, we conduct extensive experiments to assess its performance in typical tasks, including website fingerprinting and application identification. The experimental results demonstrate the effectiveness of Cactus in defending against DL-based traffic analysis attacks.

Appendix A
Effectiveness of Cactus in Defeating Rosetta

Rosetta [63] provides an approach to make DL models aware of regular changes with TCP semantics in packet length sequences, and it may significantly improve the performance of DL models on conducting traffic analysis attacks. Hence, we conduct experiments to explore whether Cactus can effectively defeat Rosetta. As Rosetta extracts robust features from flows and applies DF [53] as a representative DL model to classify traffic based on the robust features, we use DF in our experiments as well. Moreover, we conduct experiments with the dataset mentioned in Section V-A. As Rosetta is designed to enable DL models to achieve stable performance in various network environments, we train DF with Rosetta on the traffic data collected without deploying Cactus and test the model on the traffic data collected with deploying Cactus. As Figure 14 presents, Rosetta cannot accurately classify traffic when Cactus is deployed. This can be attributed to Cactus's OP-I and OP-IV operations. OP-I introduces chaff packets with varying lengths and OP-IV sets a random window size to limit the maximum size of packets, which intentionally violates the normal TCP changes that Rosetta considers.

Acknowledgment

Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSFC and other sponsors.

REFERENCES

[1] A. Acar et al., "Peek-a-boo: I see your smart home activities, even encrypted!" in *Proc. 13th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, 2020, pp. 207–218.

[2] A. Aksoy and M. H. Gunes, "Automated IoT device identification using network traffic," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.

[3] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control," IETF, USA, Tech. Rep. rfc5681, 2009.

[4] M. Backes, G. Doychev, M. Dürmuth, and B. Köpf, "Speaker recognition in encrypted voice streams," in *Proc. 15th Eur. Symp. Res. Comput. Secur. (ESORICS)*, Athens, Greece. Germany: Springer, 2010, pp. 508–523.

[5] A. Bahramali, R. Soltani, A. Houmansadr, D. Goeckel, and D. Towsley, "Practical traffic analysis attacks on secure messaging applications," in *Proc. 27th Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, San Diego, CA, USA, Feb. 2020, pp. 1–18.

[6] L. Barman et al., "PriFi: Low-latency anonymity for organizational networks," 2017, *arXiv:1710.10237*.

[7] A. Bittau, D. Giffin, M. Handley, D. Mazieres, Q. Slack, and E. Smith, "Cryptographic protection of TCP streams (tcpcrypt)," IETF, USA, Tech. Rep. rfc8548, 2019.

[8] X. Cai, R. Nithyanand, and R. Johnson, "CS-BuFLO: A congestion sensitive website fingerprinting defense," in *Proc. 13th Workshop Privacy Electron. Soc.*, Nov. 2014, pp. 121–130.

[9] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2014, pp. 227–238.

[10] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance: Website fingerprinting attacks and defenses," in *Proc. ACM Conf. Comput. Commun. Security*, 2012, pp. 605–616.

[11] J. Cao, Z. Yang, K. Sun, Q. Li, M. Xu, and P. Han, "Fingerprinting SDN applications via encrypted control traffic," in *Proc. 22nd Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*, 2019, pp. 501–515.

[12] C. Chen, D. E. Asoni, D. Barrera, G. Danezis, and A. Perrig, "HORNET: High-speed onion routing at the network layer," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 1441–1454.

[13] C. Chen, D. E. Asoni, A. Perrig, D. Barrera, G. Danezis, and C. Troncoso, "TARANET: Traffic-analysis resistant anonymity at the network layer," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Apr. 2018, pp. 137–152.

[14] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 191–206.

[15] G. Cherubin, R. Jansen, and C. Troncoso, "Online website fingerprinting: Evaluating website fingerprinting attacks on Tor in the real world," in *Proc. 31st USENIX Security Symp.*, Aug. 2022, pp. 753–770.

[16] B. Coskun and N. Memon, "Tracking encrypted VoIP calls via robust hashing of network flows," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Mar. 2010, pp. 1818–1821.

[17] T. Cui, G. Gou, G. Xiong, Z. Li, M. Cui, and C. Liu, "SiamHAN: IPv6 address correlation attacks on TLS encrypted traffic via Siamese heterogeneous graph attention network," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.)*, 2021, pp. 4329–4346.

[18] Curl. (2023). *Crul*. [Online]. Available: https://github.com/curl/curl

[19] T. Dierks and E. Rescorla, "The transport layer security (TLS) protocol version 1.2," IETF, USA, Tech. Rep. rfc5246, 2008.

[20] P. Dodia, M. AlSabah, O. Alrawi, and T. Wang, "Exposing the rat in the tunnel: Using traffic analysis for tor-based malware detection," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 875–889.

[21] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 332–346.

[22] J. Gong and T. Wang, "Zero-delay lightweight defenses against website fingerprinting," in *Proc. 29th USENIX Secur. Symp. (USENIX Security)*, 2020, pp. 717–734.

[23] R. Gonzalez, C. Soriente, and N. Laoutaris, "User profiling in the time of HTTPS," in *Proc. Internet Meas. Conf.*, 2016, pp. 373–379.

[24] Google. (2023). *Google for Developers*. [Online]. Available: https://developers.google.com/speed/docs/insights/v5/about

[25] K. Hickman and T. Elgamal, "The SSL protocol," Internet Draft RFC, Netscape, USA, Tech. Rep., 1995.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[27] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New directions in automated traffic analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 3366–3383.

[28] G. Huang et al., "Efficient and low overhead website fingerprinting attacks and defenses based on TCP/IP traffic," in *Proc. ACM Web Conf.*, 2023, pp. 1991–1999.

[29] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "Toward an efficient website fingerprinting defense," in *Proc. 21st Eur. Symp. Res. Comput. Secur. (ESORICS)*, Heraklion, Greece. Germany: Springer, 2016, pp. 27–46.

[30] M. Laštovička, S. Špaček, P. Velan, and P. Čeleda, "Using TLS fingerprints for OS identification in encrypted traffic," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2020, pp. 1–6.

[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[32] D. J. Lee, B. E. Carpenter, and N. Brownlee, "Observations of UDP to TCP ratio and port numbers," in *Proc. 5th Int. Conf. Internet Monit. Protection*, May 2010, pp. 99–104.

[33] W. Li et al., "Prism: Real-time privacy protection against temporal network traffic analyzers," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 2524–2537, 2023.

[34] X. Li, W. Chen, Q. Zhang, and L. Wu, "Building auto-encoder intrusion detection system based on random forest feature selection," *Comput. Secur.*, vol. 95, Aug. 2020, Art. no. 101851.

[35] X. Lin, G. Xiong, and G. Gou, "ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *Proc. ACM Web Conf.*, 2022, pp. 633–642.

[36] Linus Torvalds. (2023). *Linux Kernel*. [Online]. Available: https://github.com/torvalds/linux.git

[37] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "FS-Net: A flow sequence network for encrypted traffic classification," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr./May 2019, pp. 1171–1179.

[38] Y. Liu, J. Wang, J. Li, S. Niu, and H. Song, "Machine learning for the detection and identification of Internet of Things devices: A survey," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 298–320, Jan. 2022.

[39] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Comput.*, vol. 24, no. 3, pp. 1999–2012, Feb. 2020.

[40] X. Luo et al., "HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows," in *Proc. NDSS*, vol. 11, 2011, pp. 1–20.

[41] R. Meier, V. Lenders, and L. Vanbever, "ditto: WAN traffic obfuscation at line rate," in *Proc. NDSS Symp.*, 2022, pp. 1–45.

[42] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Proc. Netw. Distrib. Syst. Secur. (NDSS) Symp.*, 2018, pp. 1–15.

[43] M. Nasr, A. Bahramali, and A. Houmansadr, "Defeating DNN-based traffic analysis systems in real-time with blind adversarial perturbations," in *Proc. 30th USENIX Secur. Symp. (USENIX Security)*, 2021, pp. 2705–2722.

[44] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis, "The Loopix anonymity system," in *Proc. 26th USENIX Security Symp.*, Aug. 2017, pp. 1199–1216.

[45] Selenium Project. (2023). *Selenium Webdriver*. [Online]. Available: https://www.selenium.dev/documentation/webdriver/

[46] E. Rescorla, "HTTP over TLS," IETF, USA, Tech. Rep. rfc2818, 2000.

[47] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," IETF, USA, Tech. Rep. rfc8446, 2018.

[48] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *Proc. Netw. Distrib. Syst. Secur. (NDSS) Symp.*, 2018, pp. 1–15.

[49] M. Shen, J. Zhang, L. Zhu, K. Xu, and X. Du, "Subverting website fingerprinting defenses with robust traffic representation," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 607–624.

[50] M. Shen et al., "Real-time website fingerprinting defense via traffic cluster anonymization," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2024, p. 263.

[51] M. Shen, Y. Liu, L. Zhu, X. Du, and J. Hu, "Fine-grained webpage fingerprinting using only packet length information of encrypted traffic," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 2046–2059, 2021.

[52] M. Shen, J. Zhang, L. Zhu, K. Xu, and X. Du, "Accurate decentralized application identification via encrypted traffic analysis using graph neural networks," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 2367–2380, 2021.

[53] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Oct. 2018, pp. 1928–1943.

[54] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, "Triplet fingerprinting: More practical and portable website fingerprinting with N-shot learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1131–1148.

[55] J.-P. Smith, L. Dolfi, P. Mittal, and A. Perrig, "QCSD: A QUIC client-side website-fingerprinting defence framework," in *Proc. 31st USENIX Secur. Symp.*, 2022, pp. 771–789.

[56] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on SSH," in *Proc. 10th USENIX Secur. Symp. (USENIX Secur.)*, 2001, pp. 1–17.

[57] J. Song, C. H. Cho, and Y. Won, "Analysis of operating system identification via fingerprinting and machine learning," *Comput. Elect. Eng.*, vol. 78, pp. 1–10, Sep. 2019.

[58] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 1, pp. 63–78, Jan. 2018.

[59] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 2579–2605, 2008.

[60] T. van Ede et al., "FlowPrint: Semi-supervised mobile-app fingerprinting on encrypted network traffic," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, vol. 27, 2020, pp. 1–18.

[61] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," in *Proc. 26th USENIX Secur. Symp. (SEC)*, Aug. 2017, pp. 1375–1390.

[62] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing: An efficient defense against statistical traffic analysis," in *Proc. NDSS*, vol. 9, 2009, pp. 1–14.

[63] R. Xie et al., "Rosetta: Enabling robust TLS encrypted traffic classification in diverse network environments with TCP-aware traffic augmentation," in *Proc. 32nd USENIX Secur. Symp. (USENIX Secur.)*, 2023, pp. 625–642.

[64] J. Xing and C. Wu, "Detecting anomalies in encrypted traffic via deep dictionary learning," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Jul. 2020, pp. 734–739.

[65] T. Ylonen and C. Lonvick, "The secure shell (SSH) transport layer protocol," IETF, USA, Tech. Rep. rfc4253, 2006.

[66] W. Zhang et al., "HoMonit: Monitoring smart home apps from encrypted traffic," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 1074–1088.

Renjie Xie received the B.Eng. degree from Beijing University of Posts and Telecommunications in 2016 and the M.Sc. degree from Tsinghua University in 2019, where he is currently pursuing the Ph.D. degree. His research interests include network and machine learning security.

Jiahao Cao (Member, IEEE) received the B.Eng. degree from Beijing University of Posts and Telecommunications in 2015 and the Ph.D. degree from Tsinghua University in 2020. He was a Visiting Scholar with George Mason University. He is currently an Assistant Research Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His current research interests include network protocol security, network attack detection, and network traffic analysis.

Yuxi Zhu received the B.Eng. degree from Tsinghua University in 2022, where he is currently pursuing the Ph.D. degree. His research interests include fuzzing and fingerprinting.

Yixiang Zhang received the B.Sc. degree from Tsinghua University in 2024, where he is currently pursuing the Ph.D. degree. His research interests include network security.

Yi He received the M.Sc. and Ph.D. degrees from Tsinghua University, China. His research interests include system security and program analysis.

Hanyi Peng is currently pursuing the B.Sc. degree with Tsinghua University. His research interests include network security.

Yixiao Wang is currently pursuing the B.Sc. degree with Tsinghua University. His research interests include network security.

Mingwei Xu received the B.Sc. and Ph.D. degrees from Tsinghua University. He is currently a Full Professor with the Department of Computer Science, Tsinghua University. His research interests include computer network architecture, high-speed router architecture, and network security.

Kun Sun (Member, IEEE) received the Ph.D. degree from the Department of Computer Science, North Carolina State University. He is currently a Full Professor with the Department of Information Sciences and Technology, George Mason University. He is also the Associate Director of the Center for Secure Information Systems and the Director of the Sun Security Laboratory, George Mason University. He has more than 15 years of working experience in both the industry and academia on systems and network security.

Enhuan Dong (Member, IEEE) received the B.E. degree from Harbin Institute of Technology, Harbin, China, in 2013, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2019. He was a Visiting Ph.D. Student with the University of Göttingen from 2016 to 2017. He is currently an Assistant Research Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network security, network operations, and network transport.

Qi Li (Senior Member, IEEE) received the Ph.D. degree from Tsinghua University, Beijing, China. He is currently an Associate Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network and system security, particularly in internet and cloud security, mobile security, and big data security. He is currently an Editorial Board Member of IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING and ACM DTRAP.

Menghao Zhang (Member, IEEE) received the B.S. and Ph.D. degrees in computer science from Tsinghua University in 2016 and 2021, respectively. He is currently an Associate Professor with the School of Software, Beihang University. His research interests include programmable networks, high-performance networks, networked systems, and network security.

Jiang Li received the B.Eng. degree from Beijing University of Posts and Telecommunications in 2017 and the Ph.D. degree from Tsinghua University in 2024. He is currently an Assistant Researcher with the Zhongguancun Laboratory. His research interests include inter-domain routing security, network measurement, and AI for networking.