Article

Online Traffic Obfuscation Experimental Framework for the Smart Home Privacy Protection

Shuping Huang [†] , Jianyu Cao [*] , Ziyi Chen [†] , Qi Zhong [*] , Minghe Zhang

Posted Date: 9 July 2025

doi: 10.20944/preprints202507.0772.v1

Keywords: smart home; traffic analysis; privacy protection; traffic obfuscation; online experimental framework

Article

Online Traffic Obfuscation Experimental Framework for the Smart Home Privacy Protection

Shuping Huang [1,†], Jianyu Cao [1,2,*], Ziyi Chen [1,†], Qi Zhong [3,*] and Minghe Zhang [1]

1 School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin 541004, China
2 State Key Laboratory of Public Big Data, Guizhou University, Guiyang 550025, China
3 Faculty of Data Science, City University of Macau, Macau 999078, China
* Correspondence: jycao@guet.edu.cn, qizhong@cityu.edu.mo
† These authors contributed equally to this work.

Abstract

With the widespread adoption of smart home devices, users' home behaviors are at risk of privacy leakage due to traffic analysis attacks. Attackers can use Ethernet or WiFi sniffers to capture device traffic and identify device events based on packet length and timing characteristics, thereby inferring users' home behaviors. To address this issue, traffic obfuscation techniques have been extensively studied, with common methods including packet padding, packet segmentation, and fake traffic injection. However, existing research predominantly utilizes non-real-time traffic, such as simulated, offline, and replayed traffic, to verify whether traffic obfuscation techniques can effectively reduce the recognition rate of traffic analysis attacks on smart home devices. It often overlooks the potential impact of obfuscation operations on device connectivity and functional integrity in real network environments, which represents an inherent limitation of the experimental verification method using non-real-time traffic. To address this limitation, an online experimental framework for three fundamental traffic obfuscation techniques is proposed: packet padding, packet segmentation, and fake traffic injection. This framework ensures the continuous connectivity and functional integrity of smart home devices. The fundamental traffic obfuscation techniques implemented within the online experimental framework can only resist some of the existing traffic analysis methods; this is consistent with current research findings. Importantly, the proposed framework facilitates relevant researchers in extending these fundamental techniques to novel complex traffic obfuscation methods while continuously assessing the effectiveness of these methods online.

Keywords: smart home; traffic analysis; privacy protection; traffic obfuscation; online experimental framework

---

1. Introduction

The global smart home market continues to expand, driven by the development of the Internet of Things (IoT) and network communication technologies. According to [1], the market was valued at USD 127.80 billion in 2024 and is expected to grow at a compound annual growth rate (CAGR) of 27.0% from 2025 to 2030. However, several studies [2,3] have demonstrated that attackers can capture the network traffic of smart home devices using Ethernet or WiFi sniffers and leverage packet length and timing characteristics to infer device events, thereby analyzing users' home behaviors. For example, by monitoring the activity pattern of a smart door lock, an attacker can deduce the user's absence, while analyzing interactions with a smart camera could reveal occupancy status. Such inferences pose significant privacy risks, as attackers can reconstruct daily routines or even identify vulnerable time windows for physical intrusions.

To mitigate these risks, traffic obfuscation techniques have been proposed, typically operated between the Internet Service Provider (ISP) and the WiFi Access Point (AP), or between the AP and individual smart home devices [4]. These techniques aim to disrupt the correlation between traffic patterns and user behaviors. Current research primarily evaluates obfuscation techniques (e.g., packet segmentation and padding) using non-real-time traffic, such as simulated, offline, and replayed traffic, focusing on their ability to reduce the accuracy of traffic analysis attacks. However, as emphasized by Jmila et al. [5], such approaches fail to validate whether the obfuscation mechanisms can maintain device connectivity and functional integrity in real-world networks, since offline experiments cannot replicate dynamic network conditions like latency sensitivity or protocol compatibility. To address this limitation, this paper proposes an online traffic obfuscation experimental framework aimed at mitigating the inherent constraints of non-real-time traffic in developing and evaluating traffic obfuscation techniques. The main contributions are summarized as follows.

- An online traffic obfuscation experimental network is established, which is an operational network link between smart home devices and their external router that enables real-time capture and dynamic obfuscation of traffic patterns (including packet size and timing characteristics) while maintaining normal device operation.
- The implemented platform supports three fundamental obfuscation primitives: fake traffic injection, packet padding, and packet segmentation, and provides an extensible architecture for integrating and evaluating novel complex obfuscation methods through continuous online validation.
- Our evaluation confirms the framework's ability to preserve device connectivity and functional integrity during obfuscation. While the basic techniques demonstrate partial effectiveness against traffic analysis (consistent with existing literature), the results highlight the need for developing advanced composite methods building upon these foundational approaches to achieve stronger protection.

The remainder of the paper is organised as follows. Section 2 provides a comprehensive review of related work. Section 3 gives the online traffic obfuscation experimental network. Section 4 elaborates on the online experimental framework for three fundamental traffic obfuscation techniques. Section 5 verifies the performance of the experimental framework. Section 6 concludes this paper.

2. Related Work

Traffic obfuscation techniques have been extensively studied as a key strategy for defending against traffic analysis attacks. The core technical approaches in these studies typically include fake traffic injection, packet padding, and packet segmentation. Furthermore, the effectiveness of traffic obfuscation methods is primarily evaluated using simulated traffic, offline traffic datasets, and replayed traffic scenarios.

Simulated Traffic. In this experimental evaluation scenario, a sequence of simulated packets is constructed based on the packet sizes and timestamps of real traffic to evaluate the effectiveness of traffic obfuscation methods. For instance, Datta et al. [6] developed a Python library for traffic obfuscation by employing payload padding, packet segmentation, and random overlay during data transmission. In the experimental evaluation, the researcher extracted the payload length and sending time of real device traffic, generated simulated traffic based on these parameters, and simulated the communication between the device and the server using a socket. Subsequently, unidirectional traffic obfuscation was implemented at both ends of the socket.

Offline Traffic. In this experimental evaluation scenario, real traffic datasets, such as Packet Capture (PCAP) files, are directly modified according to predefined traffic obfuscation policies. For example, Apthorpe et al. [7] proposed the Stochastic Traffic Padding (STP) mechanism, which aims to provide a flexible balance between privacy protection and resource overhead for users. The researchers applied STP to traffic traces (in the form of PCAP files) generated by real smart home devices and assessed its effectiveness in resisting traffic analysis attacks. Wang et al. [8] proposed a traffic padding method that integrates an adaptive mechanism with differential privacy, aiming to obscure the traffic patterns of IoT devices. The obfuscated traffic traces were generated using a real-world traffic dataset collected from Amazon Echo devices and subsequently employed to evaluate the method's efficacy in countering traffic analysis attacks. Alshehri et al. [9] proposed a packet size obfuscation mechanism based on uniform noise padding, aiming to enhance privacy protection for smart home traffic in Virtual Private Network (VPN) tunnels. This method applies noise padding and encryption to packets at the smart home side to achieve packet size obfuscation, with packets subsequently being restored at the VPN receiver side. The effectiveness of the obfuscation method was validated by modifying the public dataset [10] and conducting subsequent recognition tasks. Pinheiro et al. [11] proposed an adaptive packet padding mechanism that dynamically adjusts the number of padding bytes in response to variations in home network utilization, thereby enhancing traffic obfuscation. The authors utilized a publicly available dataset [12] to extract raw traffic, adjusted packet lengths according to a predefined policy, and subsequently input the processed dataset into a classifier for evaluation, thereby assessing the effectiveness of the proposed padding strategy. Brahma et al. [13] proposed a traffic obfuscation mechanism that combines virtual packet generation with dynamic link padding. The method was experimented on based on a publicly available dataset [14], and its effectiveness in defending against traffic analysis attacks was evaluated by comparing the dataset before and after traffic shaping. Alyami et al. [15] proposed a traffic obfuscation method based on fake traffic injection. This method increases the difficulty for attackers to distinguish between devices A and B by injecting fake packets into each device according to the other's traffic pattern. The research team conducted simulation experiments using two Linux computers, synthesizing fake packets based on real traffic trajectories. Subsequently, these fake packets were integrated into the collected real traffic trajectories to validate the effectiveness of the proposed method. Zhang et al. [16] proposed a smart home traffic obfuscation method based on the virtual user technique. The method enhances privacy protection by injecting traffic fingerprints of device activities into actual traffic to obfuscate device states and user behavior. The experimental validation was conducted on an offline traffic dataset. Alyami et al. [17] proposed a method to achieve IoT traffic obfuscation by randomizing packet sizes. Instead of introducing additional noise, the method enhances traffic indecipherability by dividing Transmission Control Protocol (TCP) segments into randomly sized chunks of data to disrupt the original packet length distribution. The experimental validation relies on pre-captured interactive traffic between devices and servers with obfuscation implemented on static files.

Replay Traffic. In this experimental evaluation scenario, historical traffic is replayed with the help of tools like Tcpreplay. For example, Pinheiro et al. [18] proposed a lightweight packet length obfuscation method that combines maximum padding with random padding. This approach replays IoT traffic using Tcpreplay, obtains and pads packets via Netfilter's FORWARD hook with Socket Buffer (SKB), captures the obfuscated traffic at the interface through Tcpdump to generate PCAP files, and verifies the obfuscation effectiveness using a classifier.

The above traffic obfuscation methods were verified whether they can effectively reduce the recognition rate of traffic analysis attacks on smart home device events through simulated traffic, offline traffic and replayed traffic. In this way, traffic obfuscation cannot be assessed to determine whether it can maintain the continuous connectivity and functionality of smart home devices in the actual network environment.

To address the aforementioned issues, traffic obfuscation techniques should be implemented in the network link between the real-world servers and smart home devices. A few existing studies have attempted online verification of such methods. For example, Ibbad Hafeez et al. [19] proposed a dummy traffic generation method that transmits dummy traffic at a constant rate to the uplink during the device's inactive periods, thereby concealing the device's actual activity status. These dummy packets shares the same transmission path as legitimate traffic but are silently dropped at the destination using embedded markers that identify them as non-genuine. The performance of the obfuscation method was evaluated by conducting unidirectional injection of fake traffic in a real network environment. Similarly, Zhu et al. [20] proposed the Air-Padding method, which alters the device traffic pattern by injecting crafted packets between the AP and the device. This prevents the attacker from identifying the device type or inferring its operational state. In the experimental evaluation, the iptables tool was employed to block the device's communication and perform Air-Padding near the device using a laptop, thus interfering with air interface traffic analysis. To the best of our knowledge, while a few fake traffic injection methods have been verified using online real traffic, no studies have yet demonstrated online traffic obfuscation based on packet padding or segmentation techniques.

In summary, while existing traffic obfuscation methods have achieved certain theoretical and experimental outcomes, their performance validation has been predominantly limited to offline, non-real-time traffic scenarios. Although a handful of studies have examined fake traffic injection using real-time network traffic (e.g., [19,20]), comprehensive experimental frameworks for evaluating packet padding and segmentation techniques in online environments remain notably absent from the literature. To bridge this critical gap, this paper presents a novel online traffic obfuscation experimental framework that simultaneously supports fake traffic injection, packet padding, and segmentation techniques. Our framework addresses the fundamental limitations of current verification approaches while significantly enhancing the practical applicability of traffic obfuscation technologies in real-world smart home deployments.

3. Online Traffic Obfuscation Experimental Network

A typical smart home network structure is illustrated in Figure 1. This network comprises several key components: the Smart Home ISP, the home router, the intelligent gateway, and a variety of smart home devices. However, ordinary researchers are generally not permitted to run programs or execute scripts on these nodes. To address this limitation, we construct an experimental network on the device side of the smart home network. Specifically, we establish a programmable link between the smart home devices and the home router connected to the external network, enabling real-time obfuscation of actual traffic and validation of the effectiveness of the traffic obfuscation approach.

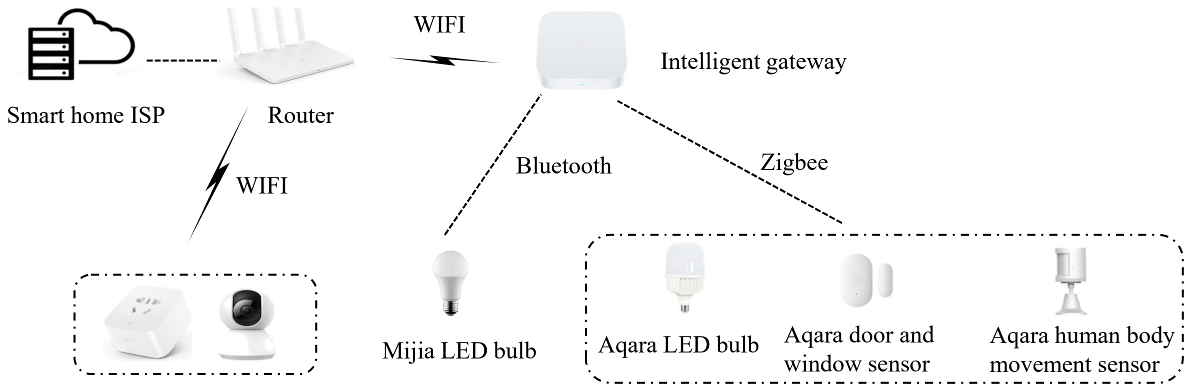

Figure 1. Smart home network structure

The structure of the experimental network is illustrated in Figure 2. The IP address, device name, and specification for each network node are detailed in Table 1. Node 1 serves as the home router, connecting to the external Internet. Nodes 2 and 7 are responsible for capturing traffic before obfuscation or after restoration. Nodes 3 and 6 process the traffic in real time based on a predefined obfuscation strategy. Node 4 is designated for capturing the traffic after obfuscation. Nodes 1–4 and 5–7 are interconnected via wired links, whereas a wireless relay establishes the connection between Node 4 and Node 5. This hybrid connectivity approach allows obfuscated traffic to traverse both wired and wireless links, thereby enhancing the realism of the experimental environment by aligning it more closely with practical smart home network scenarios.

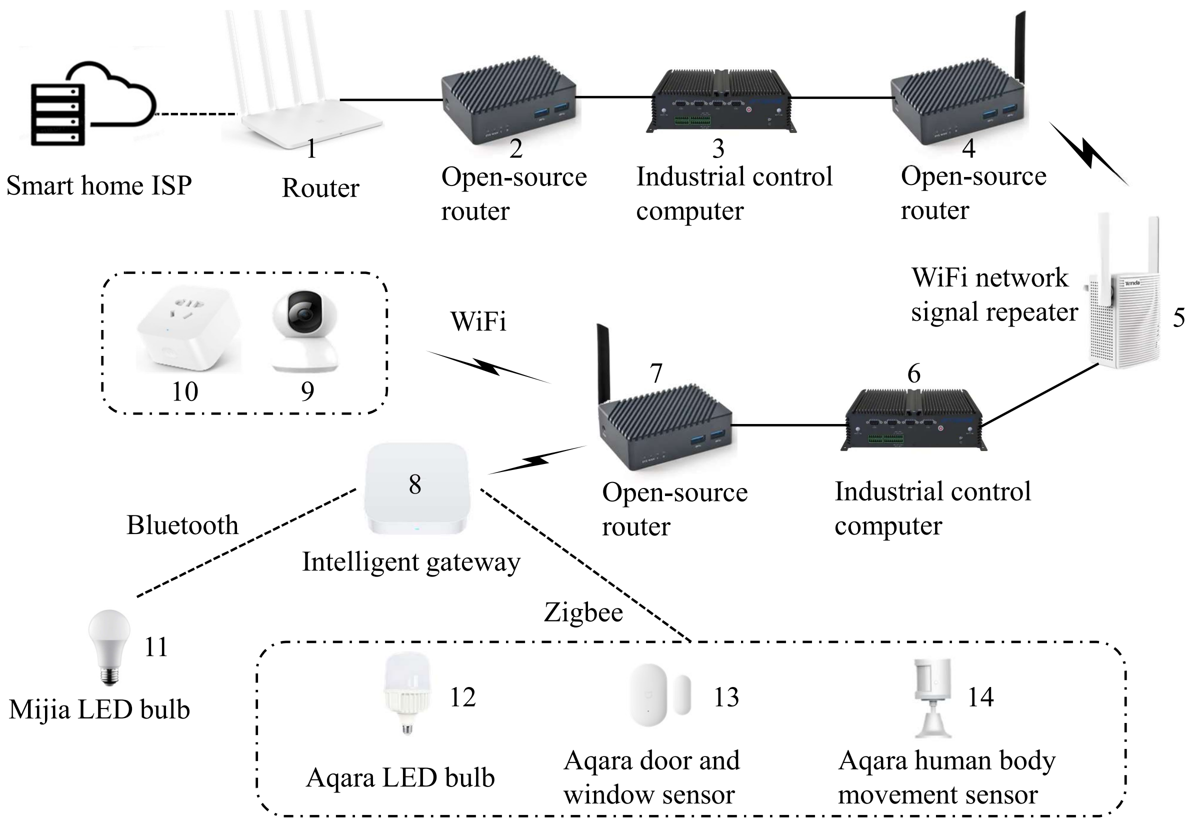

Figure 2. Online traffic obfuscation experiment network

Table 1. Specific configuration of experimental network nodes

No	IP address	Equipment name	Network mode	Specifications
1	`192.168.2.1`	Router	Ethernet	Ordinary home router
2	`192.168.2.2`	Open-source router	Ethernet	Nanopi R5S OpenWrt OS 4G Memory
3	`192.168.2.21`	Industrial control computer	Ethernet	Ubuntu OS G590-Pentium 7505 CPU DDR4 8G Memory
4	`192.168.2.3`	Open-source router	Ethernet / WiFi	Nanopi R5S OpenWrt OS 4G Memory
5	`192.168.2.4`	WiFi repeater	Ethernet / WiFi	Tenda WiFi network repeater
6	`192.168.2.22`	Industrial control computer	Ethernet	Ubuntu OS G590-Pentium 7505 CPU DDR4 8G Memory
7	`192.168.2.5`	Open-source router	Ethernet / WiFi	Nanopi R5S OpenWrt OS 4G Memory

Continued on next page

Continued from previous page

No	IP address	Equipment name	Network mode	Specifications
8	`192.168.2.175`	Intelligent gateway	WiFi / Zigbee / Bluetooth	Xiaomi intelligent multi-mode gateway
9	`192.168.2.149`	Camera	WiFi	Xiaomi intelligent camera
10	`192.168.2.204`	Smart socket	WiFi	Xiaomi smart socket
11	/	LED bulb	Bluetooth	Mijia LED bulb
12	/	LED bulb	Zigbee	Aqara LED bulb
13	/	Door/window sensor	Zigbee	Aqara door and window sensor
14	/	Human body movement sensor	Zigbee	Aqara human body movement sensor

4. Online Traffic Obfuscation Experimental Framework

Based on the experimental network presented in Figure 2, an online traffic obfuscation experimental framework has been established, as depicted in Figure 3. Its primary functions encompass original traffic capture, obfuscated traffic capture, synthetic traffic injection/termination, traffic filtering and redirection/re-forwarding, and traffic obfuscation/restoration. The original traffic capture and obfuscated traffic capture are implemented using the Tcpdump tool, and the performance of the obfuscation method is further verified through traffic analysis. Synthetic traffic injection/termination, traffic filtering and redirection/re-forwarding, and traffic obfuscation/restoration constitute the core functions of the obfuscation method, which are described as follows.

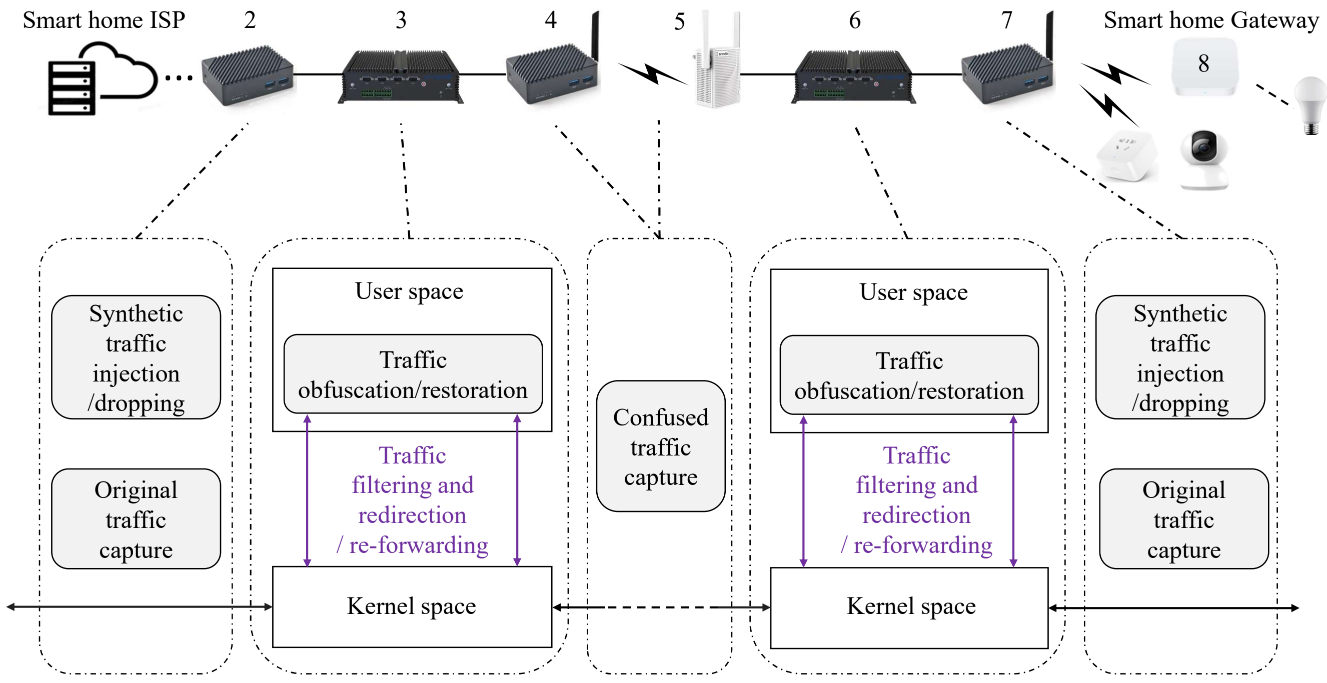
Smart home ISP
2
3
4
5
6
7
Smart home Gateway
8
Synthetic traffic injection /dropping
Original traffic capture
User space
Traffic obfuscation/restoration
Traffic filtering and redirection / re-forwarding
Kernel space
Confused traffic capture
User space
Traffic obfuscation/restoration
Traffic filtering and redirection / re-forwarding
Kernel space
Synthetic traffic injection /dropping
Original traffic capture

Figure 3. Framework for online traffic obfuscation experiments

(a) **Synthetic traffic injection/termination**

Based on the interaction traffic (captured in PCAP files) between the server and smart home devices, synthetic packets are constructed and injected from node 2 (or node 7). Subsequently, these packets assist node 3 (or node 6) in completing the injection of fake traffic, which is subsequently terminated at node 7 (or node 2).

(b) **Traffic filtering and redirection**
At nodes 3 and 6, packets are filtered and redirected based on the specified source address or destination address. For instance, the following commands are used to filter and redirect traffic associated with IP address `192.168.2.175` to `NFQUEUE` queue 1:

```
iptables -A FORWARD -s 192.168.2.175 -j NFQUEUE --queue-num 1
iptables -A FORWARD -d 192.168.2.175 -j NFQUEUE --queue-num 1
```

`NFQUEUE` is a target in the Netfilter framework that enables packets to be passed from the kernel space to user-space programs for processing. User-space programs can utilize the `NetfilterQueue` library in Python to read packets from a specified queue and determine whether to accept, drop, or alter the packets as required. The structure of these packets is illustrated in Figure 4, where the numbers in parentheses represent bit lengths. The fields highlighted in the figure indicate where modifications occur: green fields are modified during fake traffic injection, blue during packet segmentation, purple during both padding and segmentation, and yellow during all three operations. Since the data is transmitted over the Transport Layer Security (TLS) protocol, the TCP payload, also referred to as the TLS layer, consists of the "`Type`", "`Version`", "`Length`", and "`Fragment`". Here, the "`Type`" and "`Version`" fields record information about the TLS protocol, while the "`Length`" field specifies the length (in bytes) of the "`Fragment`".

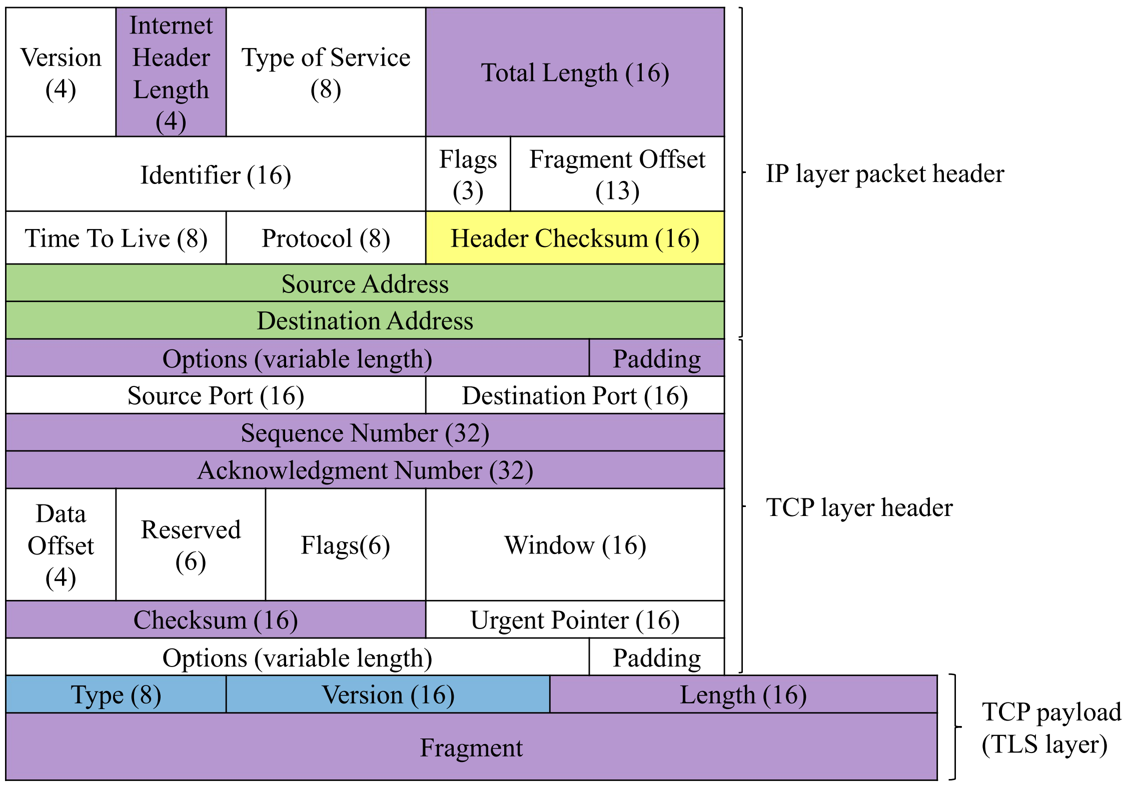

Figure 4. Packet structure

(c) **Traffic obfuscation/restoration**
At node 3, the downlink traffic (from the server to the home device) is obfuscated, and its characteristics are randomized to decrease the identifiability of specific device traffic. Prior to reaching the home device, the obfuscated downlink traffic is restored to its original content at

node 6 as required. Similarly, the uplink traffic (from the home device to the server) undergoes the same process at nodes 6 and 3, respectively, ensuring efficient bidirectional obfuscation and restoration.

(d) **Traffic re-forwarding**
After the specified traffic is obfuscated or restored in user space, it is subsequently reinjected into kernel space and re-forwarded via the `packet.accept()` method of the `NetfilterQueue` library, ensuring normal communication.

Based on the aforementioned experimental framework, this paper elaborates on the online experimental principles and execution processes of three fundamental traffic obfuscation technologies: fake traffic injection, packet padding, and packet segmentation.

4.1. Fake Traffic Injection

Fake traffic injection refers to generating a set of synthetic traffic patterns based on previously captured interaction traffic between the server and the device, and strategically injecting them into the network when the smart home device is idle, thereby effectively interfering with the attacker's judgment. The implementation principle is illustrated in Figure 5.

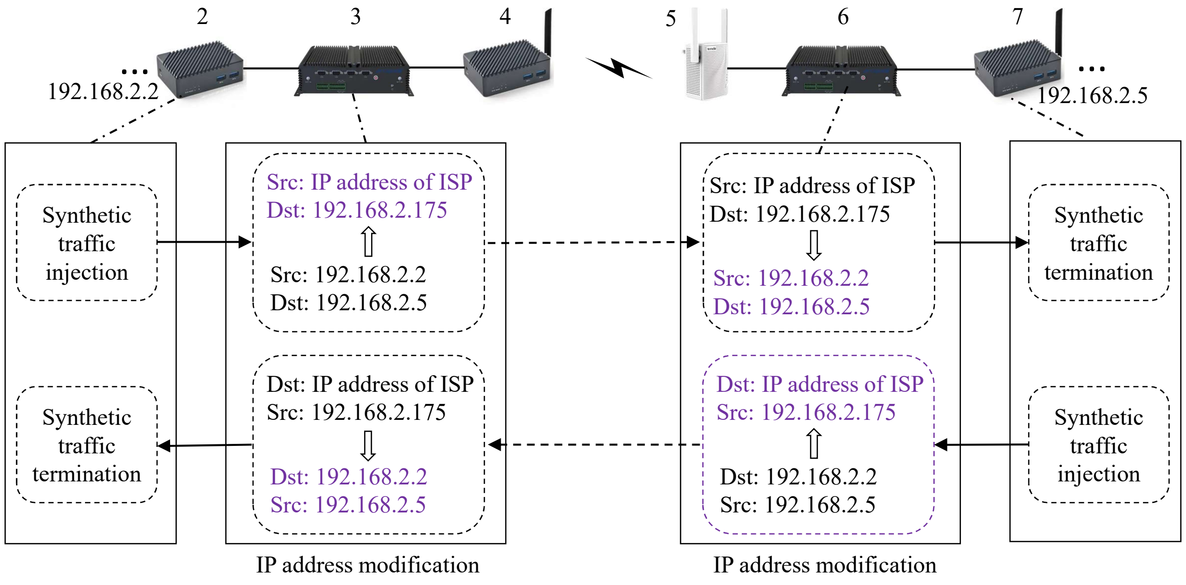

Figure 5. Implementation principle of fake traffic injection, using the smart home device with IP address `192.168.2.175` as an example

- **Synthetic traffic injection/termination**
Synthetic traffic is a foundational type of traffic specifically designed to enable controllability and emulate real device behavior, serving as a critical support mechanism for fake traffic injection strategies. On nodes 2 and 7, leveraging the captured device traffic (PCAP files), the Scapy library is employed to synthesize and inject network traffic, thereby simulating realistic bidirectional communication processes. This approach circumvents the operating system's TCP/IP protocol stack, enabling direct transmission of custom packets via Scapy's send() function, thus enhancing controllability.
In the packet injection process, various fields of IP and TCP layers can be modified flexibly as required. Additionally, packets without TCP "Reserved" field can be constructed, as this field are generally unused in the network traffic of real devices, making the structure of the generated traffic more closely aligned with actual traffic patterns. Although no actual transport-layer connection (e.g., a TCP three-way handshake) is established, key parameters such as timestamps, IP and port combinations, and sequence numbers can be utilized to reconstruct a seemingly legitimate and continuous bidirectional communication trace. The synthetic traffic constructed

on node 2 eventually reaches the destination node 7, and vice versa, with the traffic from node 7 ultimately arriving at node 2, thereby completing the termination process.

- **IP address modification**
  For downlink traffic, the process of fake traffic injection and elimination is as follows. Traffic originating from node 2 and destined for node 7 is intercepted in user space from kernel space at node 3. Subsequently, the source address of the packet is modified from `192.168.2.2` to the IP address of the smart home server, while the destination address is modified to the IP address of a device within the smart home network (e.g., `192.168.2.175`). The modified packet is then reinjected into kernel space for re-forwarding. At node 6, traffic with a source address corresponding to the smart home server IP and with a designated destination address (e.g., `192.168.2.175`) is again intercepted in user space. The source and destination addresses of the packet are subsequently restored to `192.168.2.2` and `192.168.2.5`, respectively. Finally, the modified traffic is reinjected into kernel space for further forwarding.
  For uplink traffic, similar to the aforementioned process, fake traffic injection and elimination operations are carried out at nodes 6 and 3, respectively.

4.2. Packet Padding

Packet padding refers to the process of appending random-length padding to packets during communication between the home device and the server, followed by restoring the packets to their original lengths prior to reception.

4.2.1. Implementation Principles of Packet Padding

For downlink traffic, the implementation principles of packet padding are outlined as follows.

- **Packet padding**
  At node 3, the packets destined for home devices are intercepted into the user space. The TCP payload of each packet is extended with a random-length padding. This padding is encrypted using the encryption method agreed upon between nodes 3 and 6, and the corresponding length information is stored in the IP header ”`Options`” field. The padding consists of random characters and is appended to the end of the TCP payload. Several fields are modified, including the IP ”`Internet header length`”, ”`Total length`”, ”`Header checksum`”, and ”`Options`” fields; the TCP ”`Sequence number`” (`seq`), ”`Acknowledgment number`” (`ack`), and ”`Checksum`” fields; as well as the TLS ”`Length`” field. The positions of these fields are illustrated in Figure 4. Subsequently, the padded packet is re-forwarded.
- **Packet restoration**
  At node 6, the packets destined for home devices are intercepted again into the user space. The packet is restored by removing a specific number of characters from the end of the TCP payload, where the number of characters to be removed is obtained from the IP header ”`Options`” field. The relevant fields need to be modified. Subsequently, the restored packet is re-forwarded.

For uplink traffic, similar to the downlink packet padding process, packets with a source address corresponding to home devices are padded at node 6 and restored at node 3.

In the process of padding and restoration, the fields requiring modification primarily depend on `seq`, `ack`, and TCP payload length (len), collectively referred to as the triplet $\{\texttt{seq}, \texttt{ack}, len\}$. Consider the scenario where a user sends the "turn-on" instruction to an LED bulb via a mobile terminal (as depicted in Figure 6). The update processes for `seq`, `ack`, and len are as follows. Suppose that when the "turn-on" instruction is issued, the first packet (referred to as Packet 1) sent by the smart home server to the LED bulb has a sequence number $\texttt{seq} = \alpha$, an acknowledgment number $\texttt{ack} = \beta$, and a TCP payload length $len = m$. At node 3, the packet is padded with x bytes, at which point len becomes $(m + x)$, while `seq` and `ack` remain unchanged. At node 6, Packet 1 is restored. After receiving Packet 1, the LED bulb replies with a packet (referred to as Packet 2) sent to the server, where $\texttt{seq} = \beta$, $\texttt{ack} = \alpha + m$, and $len = n$. If $n > 0$, Packet 2 is randomly padded with y bytes at node 6, in which case

len becomes $(n+y)$, ack is updated to $(\alpha+m+x)$, and seq remains unchanged. If $n=0$, Packet 2 is not padded, but ack is still updated to $(\alpha+m+x)$. At node 3, Packet 2 is restored. Upon receiving Packet 2, the server sends another packet (referred to as Packet 3) to the LED bulb, where seq $=\alpha+m$, ack $=\beta+n$, and $len=w$. If $w>0$, Packet 3 is randomly padded with z bytes at node 3, in which case len becomes $(w+z)$, seq is updated to $(\alpha+m+x)$, and ack is updated to $(\beta+n+y)$. If $w=0$, Packet 3 is not padded, but seq and ack are both updated to $(\alpha+m+x)$ and $(\beta+n+y)$, respectively. At node 6, Packet 3 is restored. The update of the triplet $\{$seq, ack, $len\}$ can be performed based on algorithms provided below.

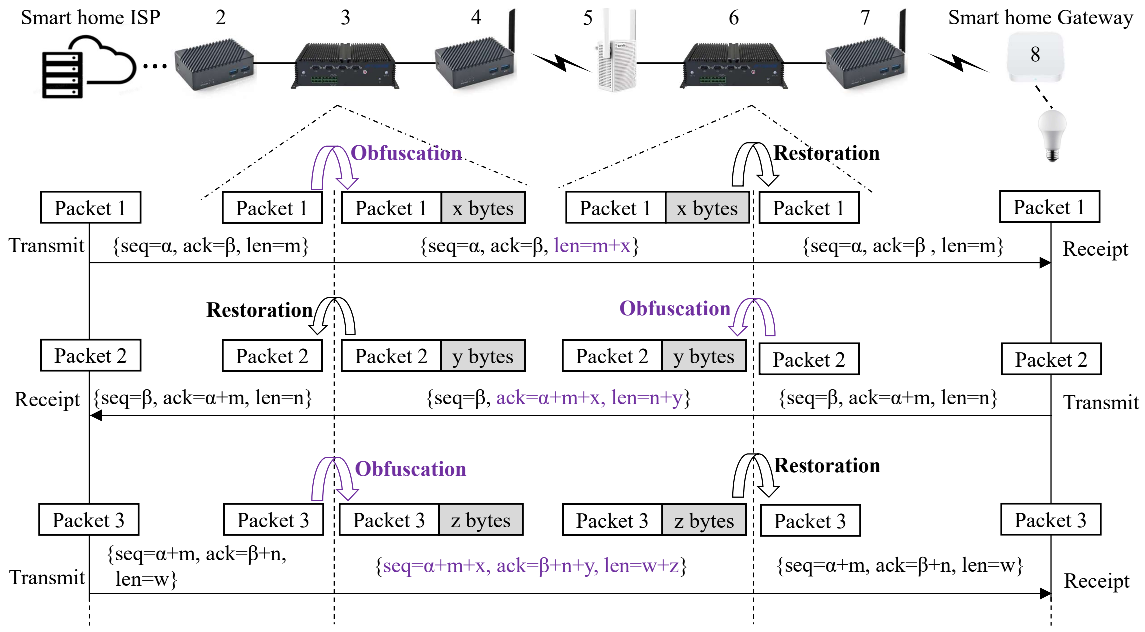

Figure 6. Illustration of the principle of packet padding and restoration

4.2.2. Key Algorithms for Packet Padding

When performing packet padding and restoration, node 3 calculates the triplet after downlink packet padding and uplink packet restoration based on Algorithm 1. In contrast, node 6 computes the triplet following downlink packet restoration and uplink packet padding, utilizing a variant of Algorithm 1.

Algorithm 1 Calculate seq, ack and len at Node 3

```
Input: IP, seq′, ack′, $len'$, $d \in \{0,1\}$, $l$, $T_0$, $T_1$
Output: seq, ack, $len$
1: $T_{src} \leftarrow T_d$,   $T_{dst} \leftarrow T_{1-d}$
2: $flag \leftarrow d$
3: $hash\_key \leftarrow$ hash(IP ‖ ack′ ‖ seq′ ‖ $flag$)
4: $E \leftarrow$ Lookup entry with key $hash\_key$ in $T_{src}$
5: if $E = \varnothing$ then
6:     seq ← seq′,   ack ← ack′
7: else
8:     seq ← $E(3)$,   ack ← $E(2)+E(4)$
9: end if
10: if $len' == 0$ then
11:     $len \leftarrow 0$
12: else
13:     $len \leftarrow len' - (-1)^d \cdot l$
14: end if
15: $new\_key \leftarrow$ hash(IP ‖ ack ‖ (seq + $len$) ‖ $(1-flag)$)
16: $T_{dst} \leftarrow T_{dst} \cup \{(new\_key$, seq′, ack′, $len')\}$
```

The inputs to Algorithm 1 include: the destination IP address (for downlink packets) or the source IP address (for uplink packets), the original sequence number $\texttt{seq}'$, acknowledgment number $\texttt{ack}'$, TCP payload length len', the packet direction flag $d \in \{0, 1\}$, the padding length l, and the global tables T_0 and T_1. The outputs are the updated values of the sequence number seq, acknowledgment number ack, and payload length len after padding or restoration. The flag $d = 1$ denotes a downlink packet, for which l bytes are padded at the end of the payload. Conversely, $d = 0$ indicates an uplink packet, from which l bytes are removed during restoration. The tables T_0 and T_1 store historical packet triplets to support sequence and acknowledgment number reconstruction. Specifically, T_1 is queried when processing downlink packets, while T_0 is used for uplink packets. The output of each transformation (padding or restoration) is recorded in the corresponding opposite table to support future reverse mapping. The hash() function is used to construct unique keys from packet metadata, and the $||$ operator denotes field concatenation. $\mathrm{E}(i)$ represents the i-th component of vector E. In Algorithm 1, the lookup table T_d is used to locate prior transformation metadata using a hash of the IP, sequence, acknowledgment, and direction flag. If no match is found, the algorithm uses the original $\texttt{seq}'$ and $\texttt{ack}'$ values; otherwise, it calculates updated values from the matched record. The payload length len is either unchanged (if $len' = 0$), or adjusted by adding or subtracting l, depending on the direction d. Finally, the current packet's metadata is recorded in the opposite table T_{1-d} to enable the future padding or restoration operation in the reverse direction.

At node 6, when calculating the triplet after padding uplink packets and restoring downlink packets, the variant of Algorithm 1 is employed. Specifically, the key differences are as follows: when $d = 1$, l bytes of data are removed from the end of the packet; conversely, when $d = 0$, l bytes of data are randomly appended to the packet. Thus, line 13 is changed to $len \leftarrow len' + (-1)^d \cdot l$.

4.3. Packet Segmentation

Packet segmentation refers to the process of segmenting and re-encapsulating the TCP payload of packets during communication between the home device and the server, followed by restoring the key fields of the packet header before the data is received. For downlink traffic, packets destined for home devices are intercepted into user space at node 3. Packet segmentation and re-encapsulation are subsequently performed in steps 1 and 2. The re-encapsulated packets are then reinjected into kernel space for forwarding. Subsequently, these packets are re-intercepted into user space at node 6, where packet restoration is executed according to step 3. Finally, the restored packets are reinjected into kernel space for further forwarding.

Step 1: TCP Payload Segmentation. If the length of a packet's TCP payload is greater than 0, the payload can be segmented into either a fixed number of segments or a random number of segments. The length of each segment is randomly determined, with the constraint that the first segment must include at least the first 5 bytes of the original packet's TCP payload (containing the "`Type`", "`Version`", and "`Length`" information of the TLS layer, as illustrated in Figure 4). In addition, selective segmentation can also be performed; that is, only a portion of packets is segmented.

Step 2: Re-encapsulation. For the first segment, first modify the first byte (the "`Type`" field of the original packet's TLS layer) to a random character of 1 byte in size, change the second and third bytes (the "`Version`" field of the original packet's TLS layer) to the length (in bytes) of the "`Fragment`" of the original packet, and modify the fourth and fifth bytes (the "`Length`" field of the original packet's TLS layer) to the length (in bytes) of the "`Fragment`" of the first segment. Then, use the original packet's TCP header and modify the "`Acknowledgment number`" (ack) and "`Checksum`" accordingly. Also, use the original packet's IP header and use the IP header "`Options`" field to mark that this re-encapsulated packet corresponds to the first segment, and this mark is encrypted with the key shared by nodes 3 and 6 to avoid being identified. Finally, modify the "`Internet header length`", "`Total length`", and "`Header checksum`" of the IP layer accordingly. For other segments, first use the original packet's TCP header and modify the "`Sequence number`" (seq) and "`Checksum`" accordingly. Additionally, utilize the original packet's IP header and utilize the IP header "`Options`" field to mark that this re-encapsulated packet is not the first segment, and this mark is encrypted with a key shared by nodes 3 and 6 to avoid detection. Finally, modify the "`Internet header length`", "`Total length`", and "`Header checksum`" of the IP layer accordingly. During the re-encapsulation process described above, the checksums for both the TCP layer and the IP layer are cleared, after which the system automatically populates them with appropriate values. The "`Internet header length`" and "`Total length`" fields in the IP layer are updated based on the specific characteristics of the re-encapsulated IP packet.

During the aforementioned encapsulation process, we obfuscate the `seq` and `ack` information of the re-encapsulated packets by delaying the transmission of TCP acknowledgment packets (with zero payload length) and adjusting their `seq` values. This approach prevents the original packet size characteristics from being inferred based on the `seq`, `ack`, and TCP payload length of the re-encapsulated packets.

Take the example of a user sending the "turn-on" instruction to an LED bulb via a mobile terminal. Suppose that upon issuing the control instruction, the smart home server transmits the first packet (denoted as Packet 1) containing application data to the LED bulb, as illustrated in Figure 7. This packet has a sequence number $\texttt{seq} = \alpha$, an acknowledgment number $\texttt{ack} = \beta$, and a TCP payload length $len = x$. At node 3, the TCP payload of this packet is divided into two segments, re-encapsulated, and subsequently re-forwarded. The first segment has a length $len = x_1$, with re-encapsulated $\texttt{seq} = \alpha$ and $\texttt{ack} = \beta$, and the second segment has a length $len = x_2$, with re-encapsulated $\texttt{seq} = \alpha + x_1$ and $\texttt{ack} = \beta$, where $x_1 + x_2 = x$. Afterward, the server receives a reply packet from the LED bulb. When the server is ready to send the next packet (denoted as Packet 3) carrying application data, it usually sends a TCP acknowledgment packet (denoted as Packet 2) first, with sequence number $\texttt{seq} = \alpha + x$, acknowledgment number $\texttt{ack} = \beta + \tau$, and TCP payload length $len = 0$. Node 3 places Packet 2 into the delayed delivery queue and temporarily refrains from forwarding it until after Packet 3 has been segmented. Let the sequence number $\texttt{seq} = \alpha + x$, acknowledgment number $\texttt{ack} = \beta + \tau$, and TCP payload length $len = y$ for Packet 3. Upon completing the segmentation of Packet 3, the first segment (Packet 3-1) is forwarded first, retaining the original sequence number `seq` while updating the acknowledgment number to $\texttt{ack} = \beta$, with TCP payload length $len = y_1$. Subsequently, the sequence number `seq` of the TCP acknowledgment packet in the delay queue is updated to $\alpha + x + y_1$, keeping the acknowledgment number `ack` unchanged, and the packet is then forwarded. Finally, for the second segment (Packet 3-2), the sequence number `seq` is adjusted to $\alpha + x + y_1$, maintaining the original acknowledgment number `ack`, where the TCP payload length $len = y_2$ and $y_1 + y_2 = y$, and it is then forwarded. If Packet 3 is randomly segmented into three or more segments, Packet 2 can be delayed and forwarded between any two segments formed by Packet 3, with the relevant `seq` and `ack` values adjusted as needed based on the actual situation.

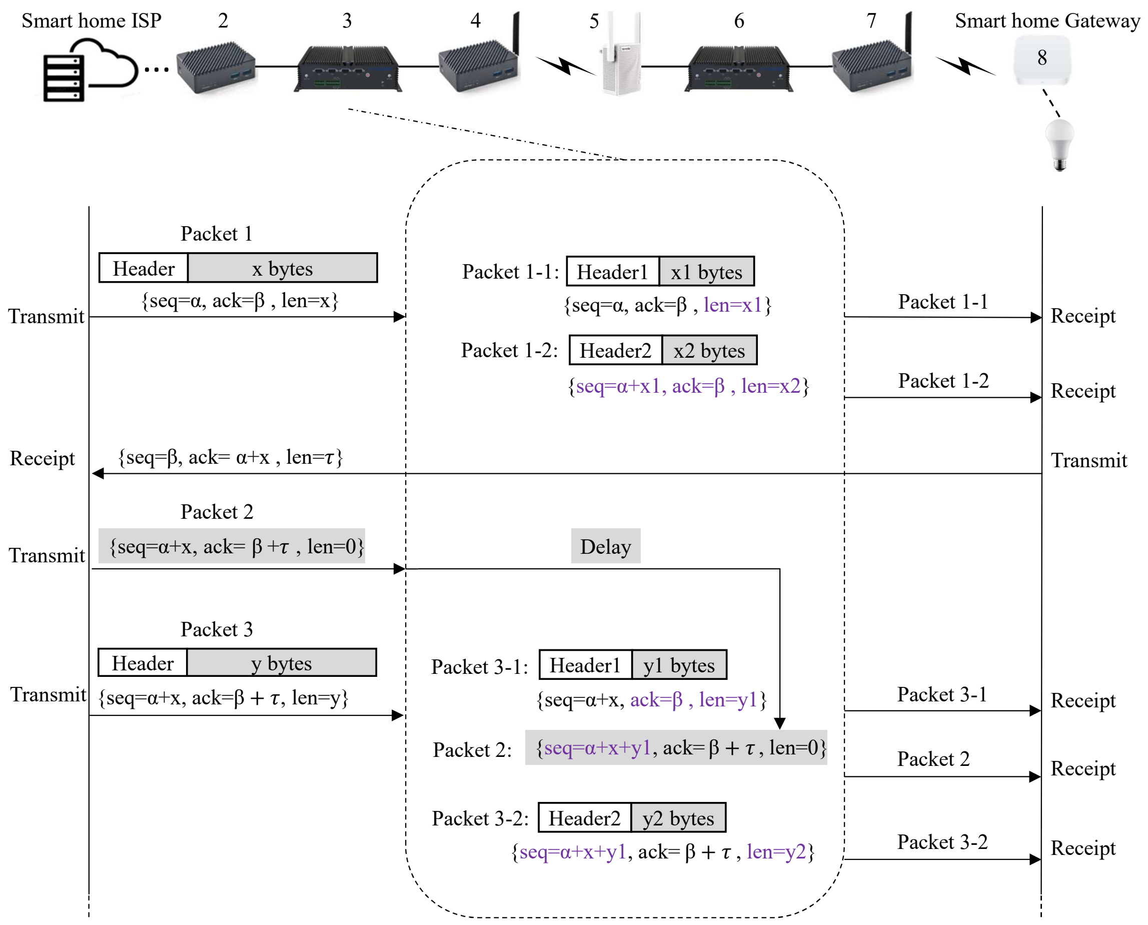

Figure 7. Example of packet segmentation with delayed redirection

The packet corresponding to the first segment is re-forwarded using the `packet.accept()` method from the `NetfilterQueue` library in Python, while the remaining packets are re-forwarded using the send() function from the Scapy library.

Step 3: Restoration. On node 6, the packet destined for the smart home device is intercepted into the user space. The packet, after being segmented and re-encapsulated, is modified again to ensure normal communication between the device and the server. For the packet corresponding to the first segment, the first 5 bytes of the TCP payload are modified by replacing the fourth and fifth bytes with the values of the third and fourth bytes, respectively. The first three bytes can be updated according to fixed characters associated with the TLS protocol. Subsequently, the IP header ”`Options`” field is removed, and the ”`Internet header length`”, ”`Total length`”, and ”`Header checksum`” are updated accordingly. For packets corresponding to other segments, their IP header ”`Options`” field is similarly removed, along with updates to their lengths and checksums. Finally, the packets are restored and forwarded as usual.

Similar to downlink packet segmentation, for uplink packets, segmentation and restoration operations are respectively performed on nodes 6 and 3 for packets with a smart home device as the source address.

5. Performance Analysis

In this section, traffic obfuscation is implemented via the proposed online experimental framework. The continuous connectivity and functionality of the device are validated to ensure system stability. Moreover, comparisons are made regarding the traffic statistical characteristics, device event recognition rate, and device recognition rate before and after obfuscation.

5.1. Continuous Connectivity and Functionality of Devices

During the experimental process, the online experimental framework continuously performed traffic obfuscation operations for over 3 hours. During this period, the smart home devices were remotely controlled via mobile terminals and consistently functioned normally, completing the on/off operations as instructed without disconnection. The experimental results confirm that the established online traffic obfuscation experimental framework effectively ensures continuous connectivity and functionality of the devices with high practicality and stability.

5.2. Traffic Statistical Characteristics

Three sets of traffic data were compared and analyzed: the unobfuscated traffic data (Not_obfuscated.pcap), the traffic data after packet padding (Packet_padding.pcap), and the traffic data after packet segmentation (Packet_segmentation.pcap). These datasets were collected during 50 switching operations performed by a mobile terminal controlling the Mijia LED bulb, with each operation separated by a uniform interval of 131 seconds. The Not_obfuscated.pcap dataset represents the original traffic captured at node 7 in Figure 3 using the Tcpdump tool. The Packet_padding.pcap dataset corresponds to the traffic that was padded and restored at nodes 3 and 6 and subsequently captured at node 4 using Tcpdump. The padding length should be randomly selected within the range of 0 to $(MSS - len)$ bytes, where MSS (Maximum Segment Size) denotes the maximum size of the TCP payload that can be transmitted in a single segment. Since the packet length in smart home devices is typically between 100 and 200 bytes, a range of 0 to 50 bytes is selected to reduce communication overhead. The Packet_segmentation.pcap dataset consists of traffic captured after segmenting TCP packets in different directions at nodes 3 and 6, with Tcpdump employed at node 4. Specifically, each packet was divided into three segments, where the TCP payload lengths of the first and second segments were randomly selected between 8 and 16 bytes, while the remaining portion served as the TCP payload for the third segment.

Compared to the unobfuscated packet length distribution, the packet padding method substantially increases the overall packet length, as evidenced by a rightward shift in the distribution. Conversely, the packet segmentation method markedly enhances the proportion of short packets, resulting in a more fragmented overall packet length distribution, as illustrated in Figure 8. These results suggest that both traffic obfuscation methods successfully alter the statistical characteristics of the original traffic.

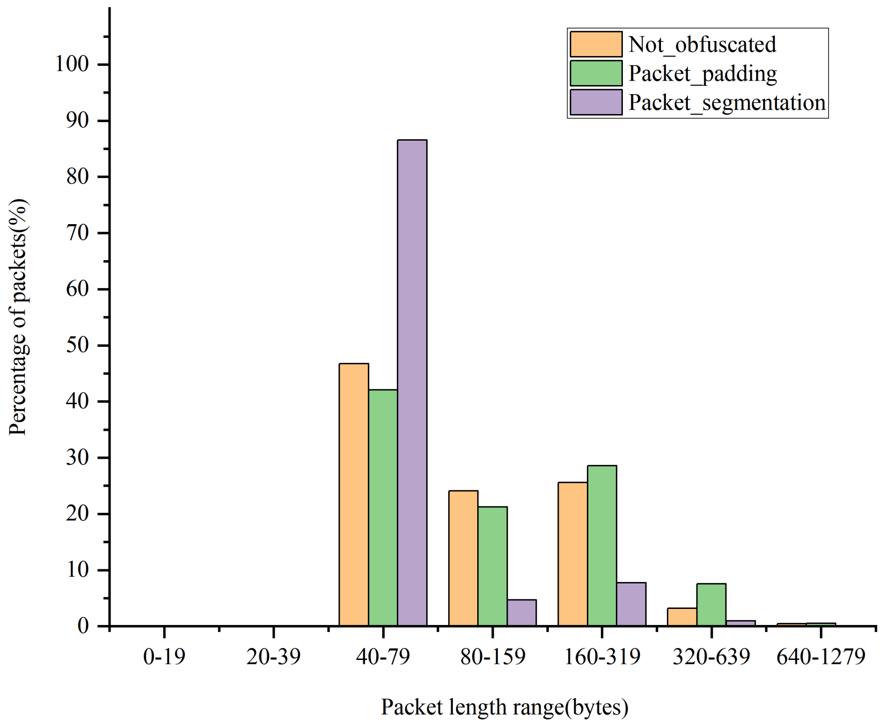

Figure 8. Packet length distribution for three different scenarios

5.3. Device Event Recognition Rate

This section verifies the recognition rate of Mijia LED bulb on/off events before and after traffic obfuscation based on the random forest classification method.

5.3.1. The Impact of False Traffic Injection on the Device Event Recognition Rate

First, the traffic data before and after fake traffic injection are captured and saved in separate PCAP files. Prior to the fake traffic injection, the Mijia LED bulb is controlled to perform on/off operations 50 times, with an operation interval of 5 seconds. This process employs the Tcpdump tool at node 4 in Figure 3 to capture mixed traffic, which includes both traffic from the Mijia LED bulb and background traffic. Subsequently, the server's traffic interacting with the Mijia LED bulb is extracted from the captured data. Based on this traffic pattern, bidirectional communication between the server and the bulb is simulated on nodes 2 and 7 in Figure 3, while fake traffic injection and elimination operations are performed on nodes 3 and 6. Obfuscated traffic is captured on node 4 using the Tcpdump tool.

Then, the on/off event fingerprints of the Mijia LED bulb, i.e., packet pairs (or sequences) of a specific length exchanged between the server and the bulbs, are extracted based on the method described in literature [21], as shown in Figure 9, and the fingerprints are saved in a CSV file.

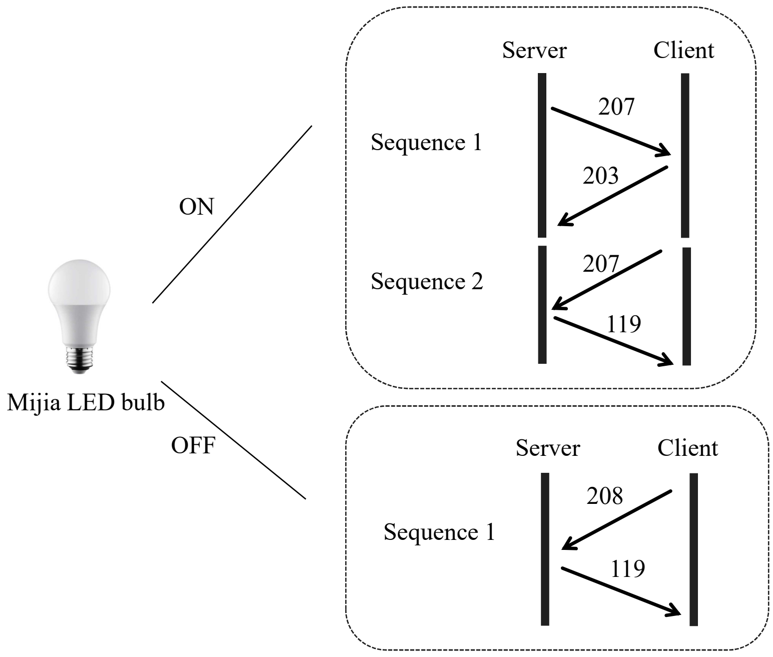

Figure 9. Mijia LED bulb on/off event fingerprints

Finally, device event recognition is performed following the process.

- Data Preparation and Model Training: The on/off event fingerprint features of the Mijia LED bulb are extracted from a CSV file. Labels are uniformly appended, and the data are merged. Additionally, the direction and event types are encoded for further processing. Subsequently, the sequence of packets within these fingerprints (comprising combinations of packet size and direction) is utilized to train a random forest classifier, enabling accurate recognition of device types and their corresponding events.
- Traffic Analysis and Device Event Recognition: The size of TCP packets and their timestamp information are extracted from the PCAP file and matched with the features of the packet sequences in the training set in terms of timing to filter out the time segments that meet the requirements. After that, the matched data sequences are predicted using the trained model to recognize the device events corresponding to them. The recognition results for Mijia LED bulb On/Off events before and after fake traffic injection are presented in Figure 10 and Figure 11, respectively.

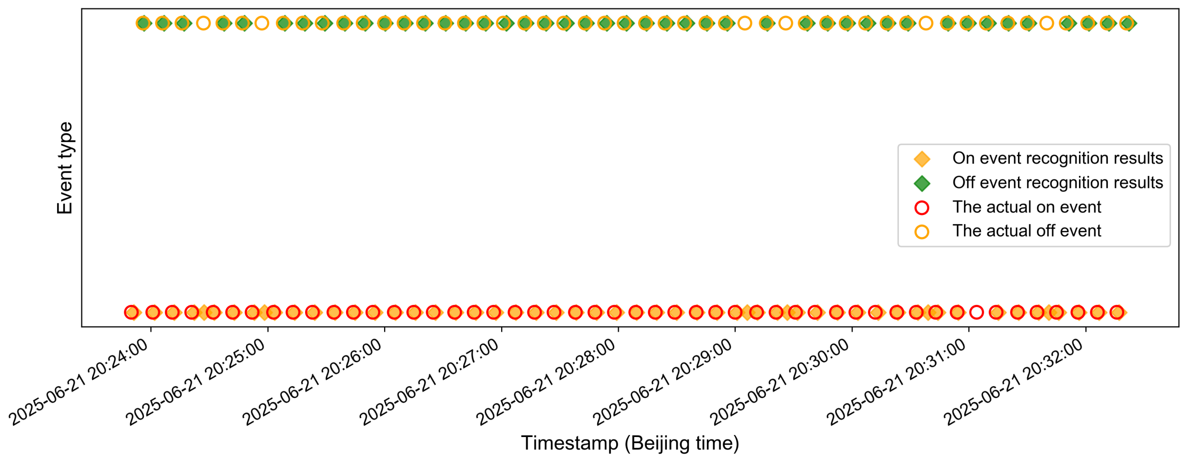

Figure 10. Recognition results for Mijia LED bulb On/Off events before fake traffic injection

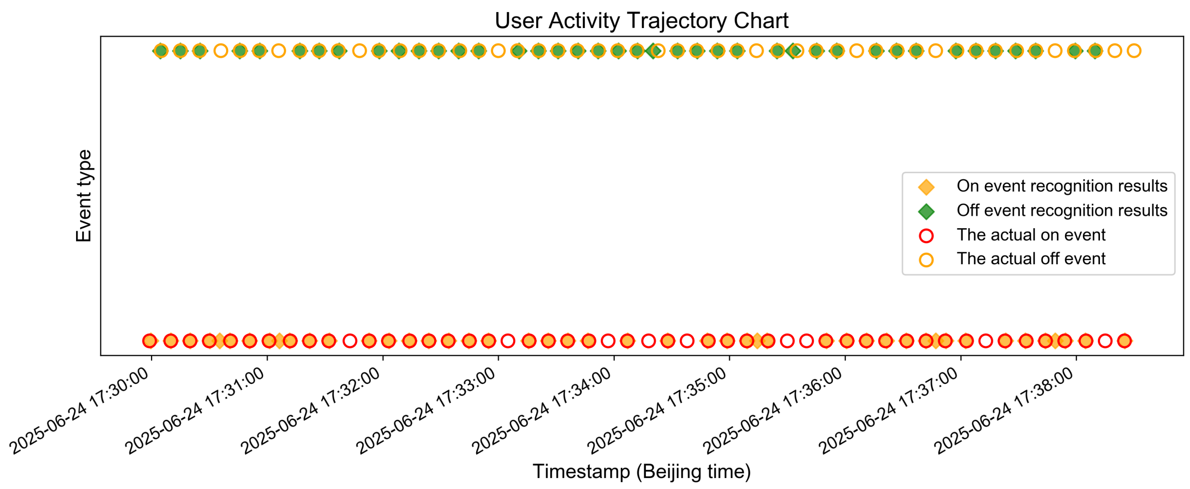

Figure 11. Recognition Results for Mijia LED bulb on/off events after fake traffic injection

After conducting a statistical analysis of the recognition results, it was found that the accuracy of Mijia LED bulb on/off event recognition before fake traffic injection was 93%. The accuracy of Mijia LED bulb on/off event recognition after fake traffic injection decreased to 79%. Fake traffic injection methods often fail to realistically replicate the complete bidirectional communication process of an IoT device, as they do not establish a genuine TCP connection, which inevitably causes packets to be out of order and thus does not exactly match the real traffic. Nevertheless, the injected fake traffic is able to obfuscate real device events with a high probability.

5.3.2. The Impact of Packet Padding and Segmentation on the Device Event Recognition Rate

The device event recognition process described above relies on the timing characteristics of the packet sequence. Packet padding and segmentation operations can have a significant impact on these timing characteristics. Specifically, packet padding changes the size of the original packet, while packet segmentation not only changes the packet size but also affects the time interval between neighboring packets. Applying these two traffic obfuscation methods will destroy the timing characteristics of the original traffic, which can significantly reduce the accuracy of this device event recognition process or even result in a recognition accuracy of 0.

5.4. Device Recognition Rate

This section investigates the impact of traffic obfuscation techniques on the recognition accuracy of Mijia LED bulbs and Xiaomi smart sockets, leveraging the deep learning methodology introduced in [22]. The training dataset collection procedure is outlined as follows: for both the Mijia LED bulb and Xiaomi smart socket, a mobile terminal executes 50 on/off operations for each device, with an interval of 5 seconds between consecutive operations. Concurrently, network traffic is captured at node 7 in Figure 3 using the Tcpdump tool. Each round of control operations (i.e., 50 on/off cycles per device) generates one PCAP file, accumulating to a total of 40 PCAP files per device, which serve as the dataset for subsequent model training. The validation datasets comprise unobfuscated traffic data, traffic data following packet padding, traffic data after packet segmentation, and traffic data from fake traffic injection. Unobfuscated traffic data is collected using the same method as the training data. Packet-padded and segmented traffic data undergo their respective obfuscation processes during collection, maintaining parameter settings consistent with those detailed in Section 5.2. For the fake traffic injection data, synthetic packets are constructed based on the original PCAP files of the Mijia LED bulb and Xiaomi smart socket, as described in Section 5.3.1.

The deep learning model introduced in [22] employs a hierarchical abstraction mechanism, enabling the transformation of original heterogeneous network traffic characteristics (e.g., packet size and time interval) into homogeneous vectors in a unified format. This input-independent model automatically extracts key features to efficiently perform traffic fingerprinting. Given the limited sample size, the K-fold cross-validation method is applied during the training phase with K set to 10, yielding 10 sub-models for evaluation. During the validation phase, for each of the two devices, a new PCAP file not used in training is selected as test data to assess the performance of each of the aforementioned 10 models.

Figure 12 demonstrates the model's accuracy in recognizing Xiaomi smart socket traffic across four scenarios (unobfuscated, packet padding, packet segmentation, and fake traffic injection). Figure 13 presents the model's accuracy in recognizing Mijia LED bulb traffic under the same conditions. Experimental results reveal that packet padding has minimal impact on the recognition performance of the deep learning model proposed in [22], with recognition accuracy remaining largely stable. In contrast, packet segmentation significantly disrupts the traffic characteristics of the Xiaomi smart socket, leading to a notable decrease in recognition accuracy. For Mijia LED bulb traffic, however, packet segmentation does not cause meaningful interference, and recognition accuracy remains consistently high. Furthermore, fake traffic injection introduces some level of obfuscation into the traffic characteristics of the Xiaomi smart socket; on average, more than 91% of samples are still correctly identified as this device—indicating that effective obfuscation is achieved. Regarding Mijia LED bulb traffic recognition tasks, however, fake traffic injection has little influence on model performance. These observations can be attributed to the binary classification setup employed in the experiments, where the model is trained to distinguish between two independently obfuscated traffic types. Since the classifier proposed in [22] is highly sensitive to class-distinguishing features, if the features of one class are obfuscated but insufficient to cause feature overlap with the other, the model still confidently identifies them as the original class. Conversely, if obfuscation causes inter-class feature resemblance misclassification can occur. This suggests that a collaborative obfuscation strategy—where obfuscation parameters are jointly set across traffic types instead of being applied individually—might enhance overall obfuscation effectiveness.

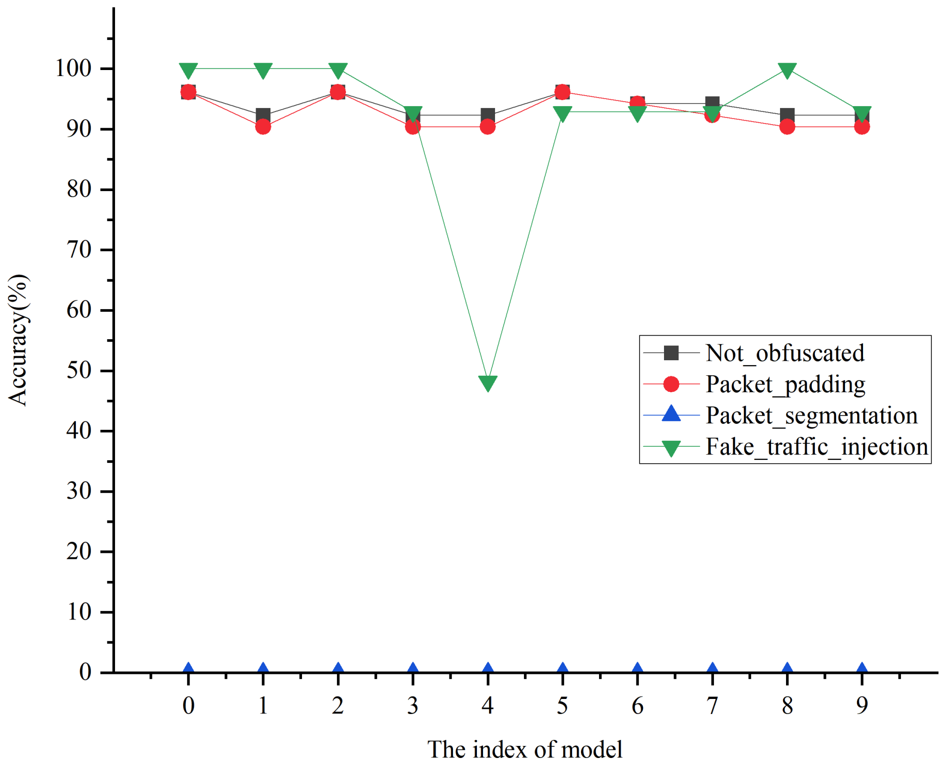

Figure 12. Recognition accuracy of Xiaomi smart socket traffic

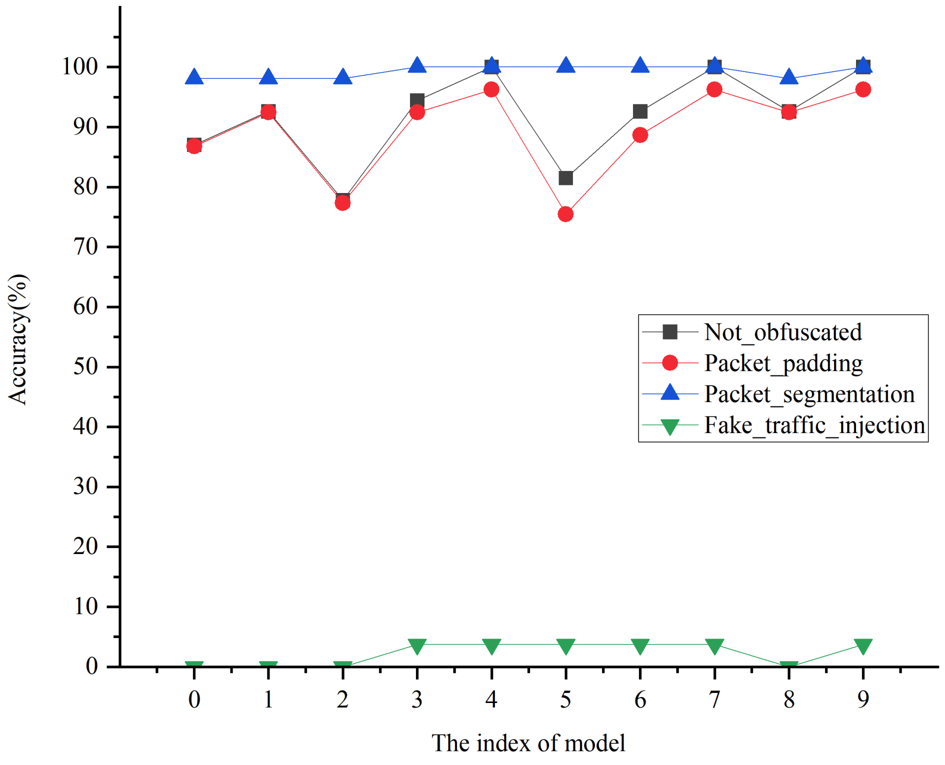

Figure 13. Recognition accuracy of Mijia LED bulb traffic

These experimental findings suggest that fundamental traffic obfuscation techniques implemented within the online experimental framework described in this paper do not fully counteract existing traffic analysis methods. This conclusion is consistent with related observations in [23], which indicate that no single obfuscation technique can resist all forms of traffic analysis. It is worth noting that the fundamental traffic obfuscation techniques implemented within the online experimental framework facilitate relevant researchers in extending them to novel complex traffic obfuscation methods while assessing the effectiveness of these methods online.

6. Conclusions

To address the limitations of non-real-time traffic in the design and performance verification of traffic obfuscation methods—particularly the inability to validate the continuous connectivity and functionality of smart home devices—this paper proposes an online traffic obfuscation experimental framework. Based on this framework, three representative traffic obfuscation techniques are implemented. These techniques employ simplified obfuscation strategies, enabling researchers to flexibly adjust the obfuscation methods and intensities according to specific requirements. Looking ahead, we expect that researchers will leverage this online experimental framework to further advance the design and experimental validation of traffic obfuscation methods.

Author Contributions: Conceptualization, S.H. and J.C.; methodology, S.H. and J.C.; software, S.H.; validation, S.H. and Z.C.; formal analysis, Q.Z.; investigation, S.H. and Z.C.; resources, M.Z.; data curation, Z.C.; writing—original draft preparation, S.H. and Z.C.; writing—review and editing, Q.Z. and M.Z.. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Natural Science Foundation of China under Grant 62361017, in part by Natural Science Foundation of Guangxi under Grant 2023GXNSFBA026212, in part by Research Foundation Ability Enhancement Project for Young and Middle aged Teachers in Guangxi Universities under Grant 2023KY0227, in part by the Open Project of State Key Laboratory of Public Big Data under Grant PBD2022-09, and in part by the National College Students Innovation and Entrepreneurship Training Program under Grant 202410595061.

Data Availability Statement: The original contributions presented in the study are included in the article, and further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Grand View Research. Smart Home Market Size, Share & Trends Analysis Report By Product (Security & Access Controls, Lighting Control), By Protocol (Wired, Wireless, Hybrid), By Application (New Construction, Retrofit), By Region, And Segment Forecasts, 2025-2030. https://www.grandviewresearch.com/industry-analysis/smart-homes-industry, 2025.
2. Skowron, M.; Janicki, A.; Mazurczyk, W. Traffic Fingerprinting Attacks on Internet of Things Using Machine Learning. *IEEE Access* **2020**, *8*, 20386–20400. DOI.10.1109/ACCESS.2020.2969015.
3. Ahsan, M.S.; Islam, M.S.; Hossain, M.S.; Das, A. Detecting Smart Home Device Activities Using Packet-Level Signatures From Encrypted Traffic. *IEEE Transactions on Dependable and Secure Computing* **2025**, *22*, 1070–1081. DOI.10.1109/TDSC.2024.3424299.
4. Apthorpe, N.; Reisman, D.; Feamster, N. Closing the Blinds: Four Strategies for Protecting Smart Home Privacy From Network Observers. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP) Workshop on Technology and Consumer Protection (ConPro '17), IEEE, San Jose, CA, USA, 25, May 2017; pp. 1–6. DOI.10.48550/arXiv.1705.06809.
5. Jmila, H.; Blanc, G.; Shahid, M.R.; Lazrag, M. A Survey of Smart Home IoT Device Classification Using Machine Learning-Based Network Traffic Analysis. *IEEE Access* **2022**, *10*, 97117–97141. https://doi.org/10.1109/ACCESS.2022.3205023.
6. Datta, T.; Apthorpe, N.; Feamster, N. A Developer-Friendly Library for Smart Home IoT Privacy-Preserving Traffic Obfuscation. In Proceedings of the 2018 ACM Special Interest Group on Data Communication (SIGCOMM) Workshop on IoT Security and Privacy (IoT S&P '18), ACM, Budapest, Hungary, 20, Aug. 2018; pp. 43–48. DOI.10.1145/3229565.3229567.
7. Apthorpe, N.; Huang, D.Y.; Reisman, D.; Narayanan, A.; Feamster, N. Keeping the Smart Home Private with Smart (er) IoT Traffic Shaping. In Proceedings of the 2017 Privacy Enhancing Technologies Symposium (PETS), Minneapolis, USA, 18-21 Jul. 2019; Vol. 2019, p. 128–148. DOI.10.2478/popets-2019-0040.
8. Wang, C.; Kennedy, S.; Li, H.; Hudson, K.; Atluri, G.; Wei, X.; Sun, W.; Wang, B. Fingerprinting Encrypted Voice Traffic on Smart Speakers with Deep Learning. In Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '20), ACM, Linz, Austria, 8-10, Jul. 2020; pp. 254–265. DOI.10.1145/3395351.3399357.
9. Alshehri, A.; Granley, J.; Yue, C. Attacking and Protecting Tunneled Traffic of Smart Home Devices. In Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy (CODASPY '20), ACM, New Orleans, LA, USA, 16-15, Mar. 2020; pp. 259–270. DOI.10.1145/3374664.3375723.
10. Sivanathan, A.; Sherratt, D.; Gharakheili, H.H.; Radford, A.; Wijenayake, C.; Vishwanath, A.; Sivaraman, V. Characterizing and Classifying IoT Traffic in Smart Cities and Campuses. In Proceedings of the 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), IEEE, Atlanta, GA, USA, 1-4, May 2017; pp. 559–564. DOI.10.1109/INFCOMW.2017.8116438.
11. Pinheiro, A.J.; de Araujo-Filho, P.F.; Bezerra, J.d.M.; Campelo, D.R. Adaptive Packet Padding Approach for Smart Home Networks: A Tradeoff Between Privacy and Performance. *IEEE Internet of Things Journal* **2021**, *8*, 3930–3938. DOI.10.1109/JIOT.2020.3025988.
12. Sivanathan, A.; Gharakheili, H.H.; Loi, F.; Radford, A.; Wijenayake, C.; Vishwanath, A.; Sivaraman, V. Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Transactions on Mobile Computing* **2019**, *18*, 1745–1759. DOI.10.1109/TMC.2018.2866249.
13. Brahma, J.; Sadhya, D. Preserving Contextual Privacy for Smart Home IoT Devices With Dynamic Traffic Shaping. *IEEE Internet of Things Journal* **2022**, *9*, 11434–11441. DOI.10.1109/JIOT.2021.3126453.
14. Ren, J.; Dubois, D.J.; Choffnes, D.; Mandalari, A.M.; Kolcun, R.; Haddadi, H. Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. In Proceedings of the ACM Internet Measurement Conference (IMC '19), ACM, Amsterdam, Netherlands, 21-23, Oct. 2019; pp. 267–279. DOI.10.1145/3355369.3355577.
15. Alyami, M.; Alkhowaiter, M.; Al Ghanim, M.; Zou, C.; Solihin, Y. MAC-Layer Traffic Shaping Defense Against WiFi Device Fingerprinting Attacks. In Proceedings of the 2022 IEEE Symposium on Computers and Communications (ISCC), IEEE, Rhodes, Greece, 30 Jun.–03 Jul. 2022; pp. 1–7. DOI.10.1109/ISCC55528.2022.9913056.

16. Zhang, S.; Shen, F.; Liu, Y.; Yang, Z.; Lv, X. A Novel Traffic Obfuscation Technology for Smart Home. *Electronics* **2023**, *12*, 3477. DOI.10.3390/electronics12163477.
17. Alyami, M.; Alghamdi, A.; Alkhowaiter, M.A.; Zou, C.; Solihin, Y. Random Segmentation: New Traffic Obfuscation against Packet-Size-Based Side-Channel Attacks. *Electronics* **2023**, *12*, 3816. DOI.10.3390/electronics12183816.
18. Pinheiro, A.J.; Bezerra, J.M.; Campelo, D.R. Packet Padding for Improving Privacy in Consumer IoT. In Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC), IEEE, Natal, Brazil, 25-28 Jun. 2018; pp. 00925–00929. DOI.10.1109/ISCC.2018.8538744.
19. Hafeez, I.; Antikainen, M.; Tarkoma, S. Protecting IoT-environments against Traffic Analysis Attacks with Traffic Morphing. In Proceedings of the 2019 IEEE international conference on pervasive computing and communications workshops (PerCom Workshops), IEEE, Kyoto, Japan, 11-15 Mar. 2019; pp. 196–201. DOI.10.1109/PERCOMW.2019.8730787.
20. Zhu, Q.; Yang, C.; Zheng, Y.; Ma, J.; Li, H.; Zhang, J.; Shao, J. Smart home: Keeping privacy based on Air-Padding. *IET Information Security* **2021**, *15*, 156–168. DOI.10.1049/ise2.12015.
21. Trimananda, R.; Varmarken, J.; Markopoulou, A.; Demsky, B. Packet-Level Signatures for Smart Home Devices. In Proceedings of the Network and Distributed Systems Security (NDSS) Symposium, San Diego, CA, USA, 23-26 Feb. 2020; pp. 1–18. DOI.10.14722/ndss.2020.24097.
22. Qu, J.; Ma, X.; Li, J.; Luo, X.; Xue, L.; Zhang, J.; Li, Z.; Feng, L.; Guan, X. An Input-Agnostic Hierarchical Deep Learning Framework for Traffic Fingerprinting. In Proceedings of the 32nd USENIX security symposium (USENIX Security 23), Anaheim, CA, USA, 9-11 Aug. 2023; pp. 589–606. https://www.usenix.org/system/files/usenixsecurity23-qu.pdf.
23. Shen, M.; Ji, K.; Gao, Z.; Li, Q.; Zhu, L.; Xu, K. Subverting Website Fingerprinting Defenses with Robust Traffic Representation. In Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA, USA, 9-11 Aug. 2023; pp. 607–624. https://www.usenix.org/system/files/usenixsecurity23-shen-meng.pdf.