WiFinger: Fingerprinting Noisy IoT Event Traffic Using Packet-level Sequence Matching

Ronghua Li*, Shinan Liu†, Haibo Hu*,Qingqing Ye*,Nick Feamster†
*The Hong Kong Polytechnic University †University of Chicago
*cory-ronghua.li@connect.polyu.hk,{haibo.hu, qqing.ye}@polyu.edu.hk
†{shinanliu, feamster}@uchicago.edu

***Abstract*—IoT environments such as smart homes are susceptible to privacy inference attacks, where attackers can analyze patterns of encrypted network traffic to infer the state of devices and even the activities of people. While most existing attacks exploit on ML techniques for discovering such traffic patterns, they underperform on wireless traffic, especially Wi-Fi, due to its heavy noisiness and packet losses of wireless sniffing. In addition, these approaches commonly target at distinguishing chunked IoT event traffic samples and they failed at effectively tracking multiple events simultaneously. In this work, we propose WiFinger, a fine-grained multi-IoT event fingerprinting approach against noisy traffic. WiFinger turns the traffic pattern classification task into a subsequence matching problem and introduces novel techniques to account for the high time complexity while maintaining high accuracy. Experiments demonstrate that WiFinger outperforms existing approaches on Wi-Fi traffic, with an average recall of 85% (v.s. 49% and 46%) and almost zero false positives for various IoT events.**

I. Introduction

IoT devices are increasingly ubiquitous in various applications, including smart homes, smart cities, and industrial automation. These devices connect to the internet for control and to transmit sensor data. However, even if the data transmitted is encrypted to prevent information leakage, existing works have demonstrated that device activities can still be inferred and exploited by passively monitoring the encrypted traffic and analyzing the patterns of traffic flows or packets [1]–[4]. Consequently, this may expose critical information, such as user behavior or the working status of security-sensitive devices, posing potential security risks for malicious actors (e.g., break-in) or leading to privacy breaches through unauthorized surveillance.

Currently, most existing benign or malicious IoT event/device fingerprinting methods [1]–[3], [5]–[7] focus on wired traffic at the TCP/IP layers, with only a few specifically studying fingerprinting on Wi-Fi traffic [8], [9]. While TCP/IP headers provide verbose information assisting accurate fingerprinting, TCP/IP sniffing poses stringent requirement on attackers [10], who must either exploit the LAN access point or obtain authorized access within the ISP's network (WAN). In contrast, since most IoT devices connect through Wi-Fi, their traffic can be easily sniffed using a wireless network adapter, significantly lowering the barrier and increasing the practicality of such attacks.

Given that network event classification has been studied for decades, an intuitive question arises: can these established solutions simply work for Wi-Fi layer attacks? Unfortunately, our findings indicate the opposite. By applying prevalent methods — both common machine learning-based and packet-matching approaches — to IoT traffic at the Wi-Fi layer, we have identified significant limitations and challenges of these solutions in tuning model performance and reducing manual efforts, as detailed in Table I.

Methods	Features	Data Vol	Label Acc	Param Tuning	Traffic Completeness
DL [11]–[13]	Raw Traffic	↑	↑	↑	–
ML [8], [9], [14]–[18]	Traffic stats	↓	↑	↑	–
Packet Match [1], [5]	Size & Dir & Time	↓	↓	↓	↑
WiFinger	Size & Dir & Time	↓	↓	↓	↓

TABLE I: Factors that influence the performance of existing approaches. ↑ indicates high dependence of performances on the factor, and vice versa. – indicates a certain level of robustness against a factor. The lower reliance, the higher feasibility and scalability.

First, the most widely adopted ML/DL-based traffic analysis approaches face limitations in adaptability [8], [11]–[13], granularity [8], [19], [20], and scalability for effective IoT event fingerprinting. **Adaptability:** These methods typically rely on handcrafted features or parameters (e.g., packet size statistics, flow duration) to classify traffic bursts or flows. Nonetheless, the most suitable feature/parameter sets vary among devices and events due to the sparseness and diversity of IoT traffic [21]. **Granularity:** ML-based methods are not fine-grained enough for event identification, particularly when differentiating events with only slight variations in packet sizes or timing. Their results can also be easily affected by heartbeat or other background traffic, which influences flow characteristics. **Scalability:** Training these data-driven models requires a large amount of labeled data from scratch, which does scale well for large IoT systems.

Second, packet-matching approaches [1], [5], albeit accurate, encounter the following two challenges on Wi-Fi traffic. **Packet Losses:** Passive Wi-Fi sniffing is susceptible to packet loss due to variable wireless channel conditions. This affects both online detection and offline training of packet-

matching approaches. **Upper-layer Variances:** Wi-Fi traffic inherits variances from upper layers, introducing significant noise. For example, Ping-Pong [1] filtered retransmission/ACK TCP packets and only focused on fingerprint TLS application packets. However, all of them become indistinguishable at the Wi-Fi layer due to WPA encryption [22], hindering packet-matching effectiveness.

This work aims to accurately fingerprint Wi-Fi IoT events by addressing the above limitations and challenges. To this end, we propose an intuitive and yet noise-agnostic packet-matching approach for wireless traffic, *WiFinger*. Inspired by the findings in [1], [5], WiFinger adopts a similar fingerprint representation: a sequences of packets with *relative timestamps*, *sizes*, and *directions*. Yet, instead of finding exact match, WiFinger's focus on detecting whether "traces" of fingerprint are present within the examined sequence. WiFinger offers three main characteristics: **(i) Adaptable:** WiFinger uses a consistent parameter setting for fingerprint extraction and matching, requiring little parameter tuning for various devices and events. **(ii) Accurate:** It is robust against Wi-Fi traffic noise and variances, and can differentiate subtle differences in packet size and inter-arrival time features distinguishing different events. Evaluation on 31 real IoT events shows that WiFinger achieves excellent performance on continuous event tracking with almost zero false positives, even including complex events like voice commands of smart speakers. **(iii) Scalable:** WiFinger fingerprints can be extracted efficiently using a few dozen of event samples, significantly reducing data collection and labeling costs. In summary, our main contributions are as follows:

- We provide insights into the differences between the Wi-Fi and TCP/IP traffic, revealing the main challenges of fingerprinting IoT event under wireless IoT traffic.
- We identify the current gap between experimental settings and online settings, showing that the performance of existing models are often exaggerated and unrealistic.
- We propose WiFinger, a fine-grained IoT event fingerprinting approach on Wi-Fi traffic, formulating the detection as a subsequence matching problem. We tackle the NP-hard efficiency challenge to ensure fast offline training and online matching while maintaining high accuracy. Moreover, we overcome the challenge of extracting fingerprints from noisy traffic using collective intelligence.

The rest of the paper is organized as follows. Section II uses real examples to show the limitations of existing methods and motivate this work. Section III defines the problem scope and threat model. Section IV introduces the intuition behind WiFinger and details its fingerprint extraction and matching process. Section V compares WiFinger to two state-of-the-art methods and investigates its performance against defenses. Section VI provides additional findings on IoT event traffic and discusses how WiFinger can be extended to other domains.

II. Related Work & Motivation

Traffic analysis and event detection have been widely studied over the last decade [1]–[3], [5], [6], [8], [23], [24]. However, most existing works struggle with fingerprinting Wi-Fi IoT events effectively, particularly compared to methods at higher layers or different protocols. This section uses real Wi-Fi event sequences to illustrate Wi-Fi IoT traffic characteristics, highlighting limitations and challenges for existing fingerprinting methods in this domain.

A. Wi-Fi Event Traffic: A Motivating Example

Using a real example, we demonstrate Wi-Fi IoT event traffic characteristics and their differences from typical TCP/IP traffic. Since Wi-Fi Control and Management packets are not related to application-layer behaviors, we only analyze Data packets with payload, similar to [1] on TCP/IP traffic. Figure 1 shows example Wi-Fi Data packet sequences from a Hue Light Bulb. Down arrows (D_{size}) represent downstream packets (router to device) and up arrows (U_{size}) represent upstream packets (device to router). Ideally, "ON" commands start with a D_{254} packet, followed by a U_{333} upstream packet 0.1 - 0.2s later, and ends with a D_{129} packet after another 0.4s. "OFF" commands are similar with "ON", except their first two packets are one-byte larger.

INSIGHT #1: Wi-Fi traffic has severe data packet loss due the nature of wireless sniffing. Though Wi-Fi includes retransmissions for station-to-AP reliability, they don't guarantee completeness for a passive sniffer in promiscuous mode. Compared to ideal event traffic, real event traffic may have missing components due to the unreliable Wi-Fi sniffing. In contrast, TCP/IP traffic sniffing via port mirroring is typically less susceptible to packet loss. According to our experiments, the best performing adapter suffers from 15-20% data packet drop, which significantly impacts the detecting performance.

INSIGHT #2: Event bursts/flows can be dominated by noise packets, obscuring actual event patterns. As illustrated in Figure 1, observed 'Real' sequences often deviate from idealized patterns, being interleaved with packets that are irrelevant with application behaviors. In contrast to TCP/IP payload analysis, Wi-Fi frames also encapsulate transport layer control packets (e.g., ACKs, SYNs), which are not directly part of the application data and ideally should be filtered. Furthermore, when IoT events occur concurrently with other background communication, their flow-level patterns of events will be mixed with, and potentially buried within, this larger volume of background noise. Consequently, event burst patterns no longer pertain to application-layer behaviors, rendering Wi-Fi flow inherently noisier than TCP/IP flows.

Based on the above insights, we further analyze some existing traffic classification approaches and why they are limited in fingerprinting wireless IoT events.

B. Flow-level Analysis

Flow-based analysis is a promising technique adopted in many network applications, e.g., general traffic analysis [25],

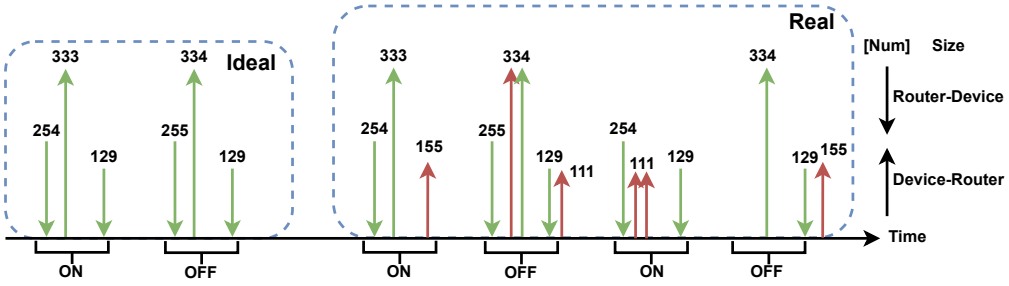


Fig. 1: An example of ideal Wi-Fi IoT event traffic v.s. real-world collected traffic. Green arrows represent fingerprint packets and red arrows are unrelated packets. While the ideal event fingerprints are clean, real-world event traffic may be incomplete or mixed with noise packets.

[26], anomaly detection [27]–[30], and some network attack detection [31]–[34]. These approaches analyze the statistical/meta data of flows (e.g., average packet sizes, inter-arrival times, flow durations, and etc.), aiming to classify events based on the similarity of repetitive flow-level features. To this end, supervised machine learning (deep learning) is usually adopted for its robustness and accuracy. While being effective in other applications, applying flow-based ML approaches to Wi-Fi IoT event fingerprinting faces significant limitations.

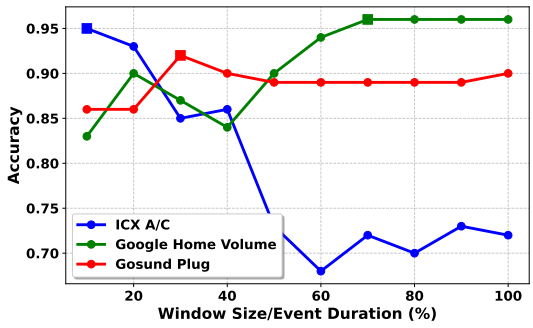


Fig. 2: [8] performance with various sliding window and devices/events using chunked training/testing samples.

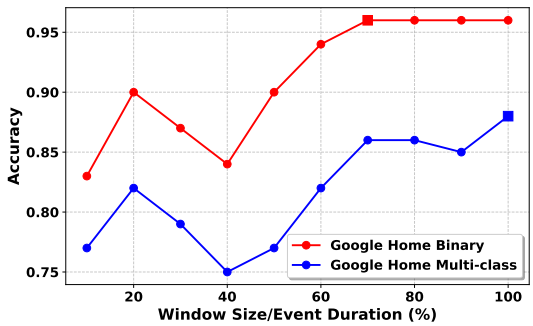


Fig. 3: [8] performance of binary classification v.s. multi-event classification using chunked training/testing samples.

LIMIT #1: Collecting labeled training data is too costly considering the countless types of devices in the market place. The limit of dataset and the time-consuming data collection process limits the use of deep learning models. DL models are notable for their capability of extracting high dimensional features automatically. However, training such models require comparable amounts of labeled data. Existing state of the art DL approaches [11]–[13] mostly focused on classification tasks with large publicly available datasets, which are yet unavailable for Wi-Fi event traffic. However, collecting hundreds or thousands of training samples per event is time-consuming (hours to days). Even worse, when a new class (event/device) is introduced, models have to be re-trained or finetuned, making the system bulky considering the rapid development of the IoT domain.

LIMIT #2: Parameter tuning and feature set selection have overly significant impact on the overall performance yet low adaptability across devices/events. For training-efficient machine learning models, expertise for feature selection and model parameters tuning is needed, which are not only labor-intensive, but also suboptimal-prune. Taking the sliding window size as an example parameter, [8] suggests setting the flow window size as a quarter of the event duration based on empirical observation. Yet, this setting hardly suits every device. We tested the influence of window size on a binary classification task among three devices[1] and noticed that the best window size ratio for different devices varies significantly, as shown in Figure 2. Considering the enormous amount of features/parameters, testing their combination and tuning parameters for optimal performance is prohibitively inefficient.

LIMIT #3: Flow-level features are too coarse-grained for multi-event classification. Some IoT events involve subtle packet size differences (e.g., single-byte variations) too granular for effective capture by aggregate flow-level features [14]–[17]. In addition, as mentioned earlier, these subtle differences can be easily obscured by flow-level noise [2], [3], [6], such as periodic device heartbeats, or by countermeasures like traffic shaping [23]. As shown in Figure 3, multi-class classification on two similar events (e.g., Google Home Volume Up vs.

[1]We obtain each device's event duration by roughly calculating the median value of all post-event periods that consist the majority of packets.

Down) has a significant performance drop compared to binary classification.

LIMIT #4: ML approaches are evaluated inappropriately. Most ML methods are overrated due to their inappropriate offline evaluation against chunked samples with accurate labels (isolated flows as samples). However, in a more realistic real-world scenario, classifiers continuously sniffs windows of streaming traffic where they may make decisions with incomplete information. Figure 4 demonstrates the bias of the two scenarios: classification performance of chunked-sample are significantly higher for Google Home events than the more realistic tracking scenario.

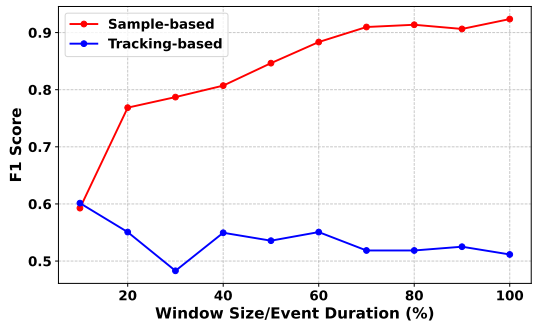


Fig. 4: F1-Score performances of chunked sample-based detection v.s. continuous tracking. Detailed introductions of "tracking" is in Section V.

LIMIT # 5: ML approaches for wireless traffic rarely handle the impact of extra noises. Some works specifically focus transferring ML approaches to fingerprint Wi-Fi traffic [8], [9], [18]. Nonetheless, they did not handle Wi-Fi specific noises, degrading the model performances on complex events or occasional background traffic bursts. Meanwhile, some works fingerprint events with other wireless protocol traffic (BLE, Zigbee, Z-Wave [35], [36]). Due to the lower traffic volume and direct data packet-to-application mapping, such protocol traffic is less noisy than Wi-Fi and thus results in better performance. Because of the identical nature of wireless traffic, we only focuses on fingerprinting the more challenging Wi-Fi traffic.

C. Packet-level Analysis

Packet-matching fingerprints events by examining sequences of packet sizes, directions, and relative timestamps [1], [5]. Ping-Pong [1] first utilized unique pairs of packet sizes, stemming from device-server request-response patterns, as event identifiers. Extending this, IoTAthena [5] incorporated timing information to form unique packet size sequences with timestamps. Both approaches demonstrated high performance in event fingerprinting, often surpassing earlier ML-based flow analysis. Unfortunately, both of them cannot operate reliably in Wi-Fi environments.

LIMIT #6: Packet-level matching approaches are vulnerable to any packet loss and the effects of upper-layer variances. The incompleteness of sniffed traffic significantly impacts online detection methods like [1], [5] due to their exact matching requirements on every single packet. Figure 5 illustrates how moderate simulated packet loss severely degrades detection performance for [1] and [5]. Furthermore, variances introduced by upper-layer protocols, combined with Wi-Fi layer properties, can interfere with their fingerprint extraction processes. Prior to extraction, Ping-Pong and IoTAthena filter for application-layer data and rely on the pairwise uniqueness of packet sizes to identify those related to specific events. However, WPA-encrypted Wi-Fi traffic prevents Ping-Pong and IoTAthena from identifying such "real" application data. In summary, packet-level losses and variations prohibit direct applications of packet-matching approaches to wireless traffic fingerprinting.

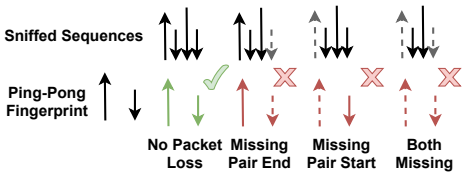


Fig. 5: Packet-level matching approaches cannot handle packet losses during sniffing.

D. Verbose Information Analysis

To avoid cumbersome ML feature engineering, some classification approaches [7], [19], [20], [37] use "verbose" raw packet information like headers or packet types (e.g., DNS, ICMP), offering two advantages: (i) reduced feature crafting effort, and (ii) greater insight into upper-layer communication protocols and interactions. For example, to avoid repetitive feature engineering, [20] directly encodes raw pcap traffic into bit sequences and deploy Autogluon [38] as a novel ML pipeline, significantly reducing the effort for data preprocessing. [7] identified that some devices always send DNS queries to specific manufactures, easily exposing their brands and types. However, for Wi-Fi layer IoT event fingerprinting, verbose upper-layer information is generally infeasible as WPA encryption renders payloads indistinguishable. Furthermore, verbose Wi-Fi frame headers are uninformative for identifying application-layer IoT events. They only reflect local network conditions rather than application behavior.

III. Problem Statement

A. Threat Model

Consistent with prior research [1], [3], [7], [8], [23], attackers aim to infer IoT device states to violate privacy by deducing living habits or to gather critical information for subsequent physical intrusions. To achieve this, attackers employ wireless sniffers operating in promiscuous mode to capture encrypted traffic. Such a sniffer could be a compromised IoT device or a self-deployed adapter outside the living space. In either scenario, attackers do not have access to the encrypted payload contents. Nonetheless, since Wi-Fi headers are in

plaintext, attackers can separate communications of different devices by their MAC addresses. Before deploying the attack, we assume attackers have a list of target devices/events and can collect labeled training samples for training. During the attack, attackers need to both identify their target devices and detect target events from a stream of traffic that involves consecutive events and idle periods.

To establish more realistic attacking scenarios, attackers identify events from streaming traffic instead of chunked samples. Such attacks are classified into three categories: naive (binary classification), single-target (targeted event detection), and multi-target (multi-event monitoring). In the naive scenario, attackers merely focus on separating a single target event from the idle state, primarily useful in limited contexts (e.g., inferring a light is "off" as the last event late at night). In the single-target scenario, attackers build fingerprints/models using multi-event training data but aim to distinguish only one specific event from all other events and idle traffic. This is helpful when targeting critical events (e.g., door unlocking) that are important/sensitive and may occur unpredictably. In the multi-target scenario, attackers aim to monitor and identify multiple target events occurring on a device. This is the most challenging case, as misclassified events can impact subsequent detection results. Despite its challenges, multi-target tracking provides the most information and is thus the ultimate goal of such privacy inference attacks.

B. Connection Configurations & Event Triggering

There are two main types of connection configurations for wireless smart devices: they either connect directly to the home router via Wi-Fi or connect to a smart hub (e.g., Amazon Echo, Google Home) which then connects to the router. Devices with the former configuration usually consume more power and often have more advanced functions. In contrast, the latter is more common for BLE or Zigbee devices that stay idle most of the time. In this work, we focus primarily on the first type of connection due to its prevalence and complexity, and our approach could be easily adapted to the hub-based configuration by sniffing communications between the hub and devices. Moreover, there are two types of event triggering schemes: via pre-configured smart home automation or via manual operations on companion apps. Since the two schemes result in similar traffic patterns [1], we focus on the latter scenario, primarily for data collection efficiency and scalability.

IV. WiFinger: System Design

In this section, we introduce our packet-level fingerprinting approach, WiFinger. Start with the analysis on Wi-Fi traffic noises, we demonstrate some intuitions on effective packet-level fingerprinting. In what follows, we first introduce what is the ideal packet-level fingerprint for noisy traffic, and how to deploy online fingerprint matching efficiently. Subsequently, we address the challenges of obtaining such ideal fingerprints.

A. Intuition on Packet-level Fingerprinting

Ping-Pong [1] initially proposed using the size of unique data packet pairs as fingerprints. IoTAthena [5] further demonstrated the significance of both packet sizes and their time intervals for TCP/IP event identification. When such data packets are transmitted over Wi-Fi, they are encapsulated within Wi-Fi data frames, adding new headers, footers, and potentially padding. Although WPA encryption hides the upper-layer packet structures, observable frame characteristics like size, direction, and timing are available. We posit that the sequence of Wi-Fi data frames corresponding to an event, despite encapsulation overheads, retains a characteristic pattern of relative timings and frame sizes derived from the original TCP/IP exchange. As a result, imagining an ideal situation with no packet loss at both layers, a Wi-Fi IoT event can still be characterized by a sequence of Wi-Fi data packets with their sizes, directions, and time intervals. We term this ideal Wi-Fi traffic sequence the ***base fingerprint*** and the constituent packets as ***fingerprint packets***. Upon an event occurrence, two key observations regarding deviations from base fingerprints are: first, occasional packet loss during an event can result in the removal of *fingerprint packets*; second, encapsulations of irrelevant packets or the variations of network environment can introduce additional packets (noise) into the observed traffic stream alongside the base fingerprint packets. Regardless of these occurrences, some or all *fingerprint packets* will remain within the event traffic, maintaining relatively stable inter-arrival times, as illustrated in Figure 6. Therefore, the event traffic classification problem can be reformulated as determining whether a given time series sequence partially matches the *base fingerprint*. In what follows, we discuss the two challenges: **1. Matching base fingerprints efficiently; 2. Extracting base fingerprints from noisy traffic**.

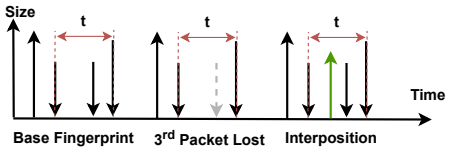


Fig. 6: Time intervals between packets remain relatively consistent when there are packet losses or interpositions.

B. Fingerprint Matching

Assuming that base fingerprints have been obtained, we conceptualize the traffic classification problem as a variation of the Longest Common Subsequence (LCS) problem. Specifically, we aim to identify similarities between the base fingerprint and a target sequence by finding a longest common subsequence, considering packet transmission directions, sizes, and interval constraints. The closer two sequences match based on this criterion, the more likely the target sequence corresponds to the same event. We define the **l**ongest **c**ommon **s**ubsequence of **n**etwork **t**raffic (NT-LCS) problem as follows:

Definition 1 (NT-LCS)**.** *For input sequences Seq_1 (base fingerprint) and Seq_2 (sequence to be examined), the NT-LCS is the longest subsequence between the two sequences:*

$$\left[p_1^a, p_2^a, \ldots, p_n^a\right], \quad p_i^a \in Seq_1^{sub}, \quad Seq_1^{sub} \subseteq Seq_1$$
$$\left[p_1^b, p_2^b, \ldots, p_n^b\right], \quad p_i^b \in Seq_2^{sub}, \quad Seq_2^{sub} \subseteq Seq_2$$

where each element p contains:

$$p_i^a = \{\, time : a_i,\ size : s_i^a,\ dir : d_i^a \,\}$$
$$p_i^b = \{\, time : b_i,\ size : s_i^b,\ dir : d_i^b \,\}$$

subject to:

- $\sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \leq \beta$ *(time constraints)*
- $(s_i^a - s_i^b)^2 \leq \epsilon^2 \quad \forall i \in [1, n]$ *(similar packet size)*
- $d_i^a = d_i^b \quad \forall i \in [1, n]$ *(same direction)*

Since the NT-LCS problem is NP-hard (proof in Appendix A), we propose a baseline approximation algorithm, FMLCS (**F**uzzily **M**atching for **L**ongest **C**ommon **S**ubsequence). FMLCS extends the dynamic programming (DP) approach for LCS, adapted to accommodate permissible variations in packet size and time intervals, as detailed in Algorithm 1. In the DP table construction, we use one table (max_{tab}) to track common subsequence lengths and another ($lcss_{tab}$) to record possible subsequences corresponding to the lengths at each step. During the DP function, two packets are matched if they share the same transmission direction and their size difference is less than or equal to ϵ bytes (line 30). After identifying potential longest common subsequences (line 6-9), their temporal alignments with the *base fingerprint* are measured by calculating the L2Norm distances between the relative timestamps of the matched packets (line 17-24). The subsequence with the best temporal alignment (minimum distance) is selected as the matching result (line 10-12). For a match to be deemed successful, the selected subsequence's length must be at least $\gamma\%$ of the base fingerprint's length and exhibit strong temporal alignment ($dist \leq \beta$). During experiments, we set $\epsilon = 1$, $\gamma = 0.6$ (60% of the base fingerprint), and $\beta = 2$. These values serve as the thresholds for successful matching (line 13).

For smart devices with simple commands (e.g., plugs, light, A/C controller), FMLCS is already sufficient to detect IoT events accurately and efficiently. However, for complex devices with larger volume of traffic (e.g., Amazon Echo Dot), this baseline has two severe limitations. **Computation efficiency:** The algorithm is computationally expensive as the time complexity of DP increases with target sequence length. Furthermore, obtaining all possible longest common subsequences requires either recording all matches during DP (current implementation) or recursive backtracking (even slower); both consume unacceptable time for long target sequences. **Accidental mismatch:** In rare cases, IoT events can be mismatched. This occurs because FMLCS only checks temporal alignment for the longest common subsequences. If a *fingerprint packet* is not event-unique, accidentally matched noise packets can extend the maximum length of common subsequences and consequently impact the final temporal alignment, as shown in Figure 7. To tackle these problems, we further analyze the characteristics of complex event fingerprints and propose a more advanced approximation algorithm.

Algorithm 1 Baseline FMLCS

1: **Input:** Seq_1 (base fingerprint), Seq_2 (target sequence)
2: **Output:** $true/false$ (whether Seq_2 matches Seq_1)
3: **function** FMLCS(Seq_1, Seq_2)
4: $max_{len}, lcss_{tab}$ = $DP(Seq_1, Seq_2)$
5: $dist_{min}$ = ∞, Seq_{match} = []
6: **for** $lcs \in lcss_{tab}$ **do**
7: **if** $len(lcs) < max_{len}$ **then**
8: **continue**
9: $dist_{temporal}$ = $TimeAlignment(lcs)$
10: **if** $dist_{temporal} < dist_{min}$ **then**
11: $dist_{min}$ = $dist_{temporal}$
12: Seq_{match} = lcs
13: **if** $len(Seq_{match}) \geq \gamma * len(Seq_1)$ & $dist_{min} < \beta$ **then**
14: **return** $true$
15: **else**
16: **return** $false$
17: **function** TIMEALIGNMENT(lcs)
18: $Subseq_1$ = $Seq_1[index[1]$ for $index$ in $lcs]$
19: $Subseq_2$ = $Seq_2[index[2]$ for $index$ in $lcs]$
20: $time\_vec_1$ = $[packet[time]$ for $packet$ in $Subseq_1]$
21: $time\_vec_2$ = $[packet[time]$ for $packet$ in $Subseq_2]$
22: $time_1$ = $time\_vec_1 - mean(time\_vec_1)$
23: $time_2$ = $time\_vec_2 - mean(time\_vec_2)$
24: **return** $l2norm(time_1 - time_2)$
25: **function** DP(seq_1, seq_2)
26: $paths$ = Empty list
27: $max_{tab} = [[0] * len(seq_1)] * len(seq_2)$
28: $lcss_{tab} = [[paths] * len(seq_1)] * len(seq_2)$
29: **for** i=1 to $len(seq_1)$ **do**
30: **for** j=1 to $len(seq_2)$ **do**
31: **if** $seq_1[i] \approx seq_2[j]$ **then** # match "common" packets
32: $max_{tab}[i][j] = max_{tab}[i-1][j-1] + 1$
33: $path_{all} = lcss_{tab}[i-1][j-1]$
34: **for** $path$ in $path_{all}$ **do**
35: $path_{all}.append([(i, j)])$
36: $lcss_{tab}[i][j]$ = $path_{all}$
37: **else**
38: **if** $max_{tab}[i-1][j] > max_{tab}[i][j-1]$ **then**
39: $max_{tab}[i][j]$ = $max_{tab}[i-1][j]$
40: $lcss_{tab}[i][j]$ = $lcss_{tab}[i-1][j]$
41: **else if** $max_{tab}[i-1][j] < max_{tab}[i][j-1]$ **then**
42: $max_{tab}[i][j]$ = $max_{tab}[i][j-1]$
43: $lcss_{tab}[i][j]$ = $lcss_{tab}[i][j-1]$
44: **else**
45: $max_{tab}[i][j]$ = $max_{tab}[i-1][j]$
46: $lcss_{tab}[i][j]$ = $lcss_{tab}[i-1][j] + lcss_{tab}[i][j-1]$
47: **return** $max_{tab}[-1][-1], lcss_{tab}$

C. Advanced Fingerprint Matching

Through careful analysis, we found that the above challenges stem from the same reason: the current common packet-matching step (line 30) considers only sizes and directions, not time. Consequently, if many packets have similar sizes and directions but different timestamps, the number of sub-

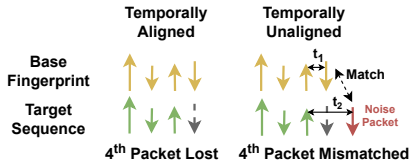


Fig. 7: Erroneous packet-matching predicated only on sizes and directions, leading to temporal unalignment.

sequence combinations increases exponentially, even resulting in erroneously matching of temporally unaligned packets. To address this issue, we propose two key optimizations to fully leverage temporal information: Anchor Reference and Fingerprint Segmentation. **Anchor Reference:** Given an IoT event, intervals between fingerprint packets remain relatively steady. This interval steadiness can be utilized to filter unnecessary packet-matching during DP, as shown in Figure 8. Specifically, we start by finding the first packet match based only on sizes and directions. Once matched, these two packets serve as anchor packets for all subsequent potential matches by constraining their time intervals relative to the anchor packets, i.e., $abs(t_1 - t_2) \leq \alpha$. Considering network fluctuation, we empirically set α to 0.2s to accommodate regular network jitters. **Fingerprint Segmentation:** For high-traffic devices,

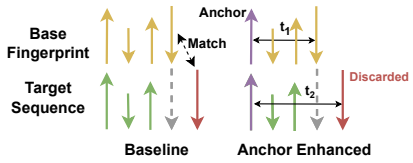


Fig. 8: Using anchor packets to filter out erroneous packet-matching, i.e., $abs(t_1 - t_2) > \alpha$.

we additionally introduce *segmented matching* based on a critical observation: packets within long base fingerprints exhibit temporal clustering patterns at finer time scales. This stems from complex IoT events involving multi-round communications, manifesting as small packet bursts (segments). Therefore, we decompose the full-sequence matching task into potentially parallelizable subtasks, each targeting one segment of the *base fingerprint*. To achieve this, we divide the base fingerprint using consecutive packet intervals, with boundaries set at the middle of intervals exceeding a threshold of 0.5s. Segments containing fewer than three packets are merged into subsequent segments. Finally, we apply the anchor-constrained matching FMLCS to all segments independently and aggregate the matching results (Figure 9). This approach effectively mitigates the combinatorial explosion in DP-based packet matching that scales exponentially with sequence length. Overall, the advanced FMLCS (AFMLCS) works as follows:

- **1. Fingerprint segmentation:** A long base fingerprint is divided into several segments.

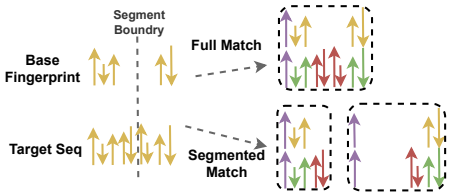


Fig. 9: Segmentation reduces the number of processed packet in each segment, decreasing the computation costs.

- **2. Anchor packet selection:** WiFinger sequentially compares fingerprint packets within the first segment with the target sequence to identify the first matching packet pair as anchors. If no such match is found, the remaining matching process is terminated.
- **3. Match the first segment:** The anchor-constrained FMLCS is applied to the first segment and the target sequence. If this matching fails, matching for the remaining segments is terminated.
- **4. Merge segmented matching results:** After a successful match for the first segment, the same anchor-constrained matching is applied to all remaining segments, whose results will get merged. Such a final result is successful if it involves at least $\gamma\%$ of the base fingerprint packets and they align well temporally (e.g., meeting the β distance criterion).

D. Fingerprint Extraction

While the matching approach relies on accurate *base fingerprints* for detection, **acquiring clean and reliable ones presents fundamental challenges**: ideal base fingerprints need to be obtained from noisy training data exhibiting high packet losses (5-20% observed rates) and upper-layer variations. To address this, our solution exploits the *collective intelligence* of repeated event executions: each noisy event traffic burst is assumed to partially manifest the base fingerprint sequence. Building upon this idea, we formulate base fingerprint extraction as another subsequence matching problem across all recorded event traffic bursts. Unlike the online matching paradigm, subsequence matching is used differently during extraction. We first use FMLCS to extract pairwise common subsequences among all corrupted traffic bursts to obtain a set of potential fingerprint components. Then, we merge these components into one noisy coarse fingerprint (CF) and leverage statistics of packet matching frequencies to refine the CF into the base fingerprint. Overall, the fingerprint extraction process consists of three steps: (i) data collection and filtering; (ii) coarse fingerprint construction; and (iii) fingerprint refinement, as illustrated in Figure 10.

1) Data Collection, Filtering, and Compression: To obtain the training dataset, we use ADB (Android Debug Bridge) [39] and Python scripts to build an automatic event execution tool, simulating taps/slides on mobile devices [1]. The tool initiates a specific event X times (30 in experiments) to trigger traffic generation from devices/servers. During automated data

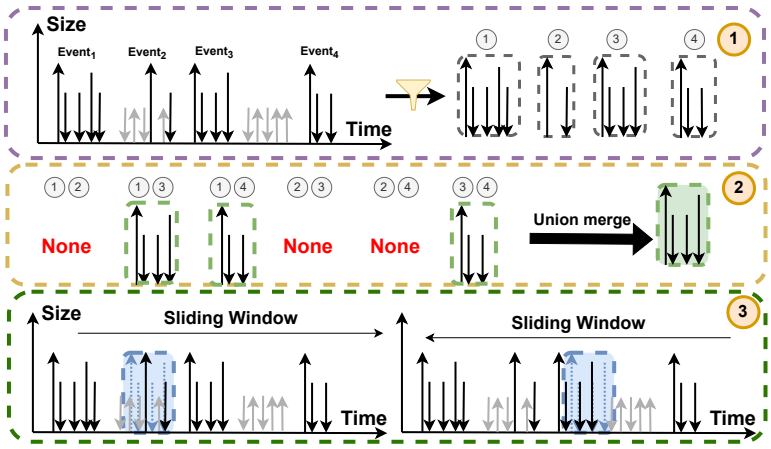


Fig. 10: Fingerprint extraction process. Step 1 filters out noise packet and cluster packets into groups. Step 2 applies pairwise AFMLCS to the groups and merge their consensus subsequences. Step 3 refines the coarse fingerprints by measuring packets' matching frequencies against the training data.

collection, the tool records timestamps of event initiations (taps/slides) and captures the Wi-Fi packets exchanged between the device and the router. Random gaps of 30-45 seconds are introduced between each event to ensure the device finishes the event and returns to the idle state.

After data collection, we first discard Wi-Fi Management and Control packets, focusing only on Wi-Fi Data packets carrying encrypted payload, as discussed in Section II-A. To avoid collecting repeated data packets of Wi-Fi retransmissions, we only keep the original packet or one of its retransmitted version if the original one is not sniffed. For high traffic devices, we further filter irrelevant noise to reduce computation costs with the remaining data packets. First, packets are categorized into classes by their sizes and directions, and those classes whose frequencies are significantly higher than the average frequency (e.g., exceeding it by 2σ or more) are discarded. These exceptionally frequent classes likely correspond to background noise, such as TCP ACK packets and regular data uploads. Then, we compress consecutive packets with the same sizes and directions into one packet. This is because identical packets mainly cause the combinatorial explosion challenge in FMLCS, but they do not contribute to the characteristics of request-response patterns of IoT events [1]. As a result, this compression significantly reduces the number of packets to be processed with limited impact on performance. For example, in our experiments, the number of training packets for Amazon Echo Dot is reduced from 10000 to 3000 after compression. Finally, according to [1], IoT events typically last no more than 10 seconds. Therefore, packets sent beyond a 15-second window following event initiations are discarded. In the end, we obtain X groups of 15-second packet traces, each corresponding to an event.

2) Coarse Fingerprint Extraction: We use the X groups of preprocessed traffic to construct a coarse fingerprint (CF). A CF is expected to contain the base fingerprint but may also include some noise packets. To extract a CF, we leverage the *collective intelligence* of traffic groups: every group of event traffic is assumed to partially embody the base fingerprint. As a result, we extract consensus subsequences from every pair of traffic groups and take the union of these consensus subsequences as the CF. To this end, we adopt an adapted AFMLCS for the extraction process.

Adaptations: The key difference between the matching and extraction phases is the noisiness of the two sequences being compared. In the matching phase, the base fingerprint is assumed to be noise-free. Thus, the naive anchor packet selection (based only on packet sizes and directions) does not severely sacrifice efficiency or accuracy. However, when both sequences are noisy, selected anchors for both sequences are very likely to be noise packets, leading to erroneous matches for all subsequent packets. Even worse, during the extraction, such mismatched subsequences (full of noise) will also be treated as potential fingerprint components, consequently resulting in significantly increment of AFMLCS's computational cost and introducing considerable noise into the final CF.

To address this, we insert fake packets at the event initiation timestamps to serve as robust anchor packets. Thanks to the stable network connection of the attackers' testbed, traffic bursts between IoT devices and routers always emerges with a fixed delay after event initiations. Since the intervals between fingerprint packets inside the burst are also steady, event initiation timestamps serve effectively as reference anchors in AFMLCS. As such, we insert fake anchor packets with a consistent size and direction at each event initiation timestamp and align sequences temporally with the new anchors, as shown in Figure 11. After obtaining all pairwise common subsequences, we naively merge the packets from these pairwisely matched subsequences into a single coarse fingerprint (CF).

3) Fingerprint Refinement: Owing to the rudimentary union merging, CF inadvertently incorporates redundant noise pack-

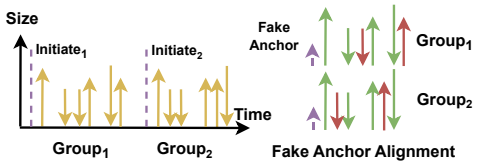


Fig. 11: Align traffic groups by their event-initiation timestamps (fake anchor packets) during the extraction.

ets. To filter such noises, we refine the fingerprint by conducting AFMLCS against the all groups using a sliding window scheme, while meticulously recording the matching frequencies for the packets inside CF. This aims to effectively pinpoint CF packets whose matching frequencies approximately align with the number of events, i.e., X. While a unidirectional sliding window might result in elevated matching frequencies for packets located at the beginning or end of the sequence, we execute the sliding window analysis in both directions. In our bidirectional refinement, packets with matching frequencies less than X are discarded. We iteratively execute the refinement process until no more packet is discarded, and construct the base fingerprint with all retained packets.

E. System Workflow

In general, the workflow for WiFinger is shown in Figure 12. During the extraction phase, WiFinger uses the event triggering module to trigger each event 30 times, collects the corresponding Wi-Fi traffic, and extracts base fingerprints for the events. In the online detection phase, WiFinger selects the target base fingerprint and match it to newly sniffed device traffic using either FMLCS or AFMLCS, depending on the traffic volume and the fingerprint length.

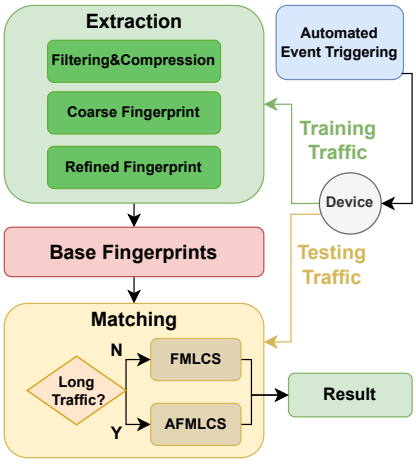


Fig. 12: WiFinger system workflow.

V. Evaluation

We conduct extensive experiments corresponding to the three attacking scenarios to evaluate WiFinger's performance against two state-of-the-art ML-based fingerprinting methods applied to Wi-Fi traffic: Peek-a-boo [8] and IoTBeholder [9]. For Peek-a-boo, we use the same feature set and select their best-performing model (Random Forest, or RF) for evaluation. **It is worth noting that we could not run Ping-Pong and IoTAthena as comparison baselines due to their incapability of handling packet losses during the experiments**.

During the experiments, we assume RF and IoTBeholder have finished their device classification and only focus on the event detection. As for WiFinger, we aim to use a single fingerprint to classify devices and events at the same time.

A. Dynamic Tracking & Evaluation Metrics

We use continuous event tracking to emulate realistic attacks instead of classifying chunked traffic flow samples. To this end, every detection method uses a sliding window to dynamically select a group of most recent packets for classification. For Peek-a-boo, we test various window sizes and select the best-performing setting for each event. For IoTBeholder, the window size is set as the burst duration, using the same definition as [9]. For WiFinger, the window size is set as the duration of the extracted base fingerprint plus two seconds (to accommodate potential timing variations). Given a window of packets, each method determines whether it corresponds to "idle" (negative) or an event (positive). Whenever an event is detected, packets within the current window are excluded from subsequent windows to avoid misclassification of similar events. Specifically, Peek-a-boo and IoTBeholder skip all packets in the next 6 seconds, while WiFinger skips all packets contained within the current window. This setting is practical as IoT events on the same device seldom occur consecutively within a very short period, and the collected events in our dataset have long enough gaps in between.

To evaluate dynamic tracking performance, we use precision and recall rates as metrics. During detection, true positive results correspond to events with correct labels, and false positive results correspond to events with wrong labels or misclassifications of idle states. Precision is defined as the ratio of true positive detections to the total number of positive detections. Recall is defined as the ratio of true positive detections to the total number of triggered events.

B. Attacking Scenarios

We evaluate methods' dynamic tracking performance in three scenarios: naive, single-target, and multi-target tracking (ultimate objective).[2] For Peek-a-boo and IoTBeholder, attackers use various strategies to train models and detect events to achieve the optimal performance.

- **Naive:** attackers train simple binary classifiers to distinguish one target event from idle periods.

[2]It is worth noting that some event fingerprints (E11-E13, E16-17) are inseparable, they excluded from the single-target/multi-target experiments.

- **Single-target:** attackers train multi-event classifiers for each device and aim to distinguish only the event of interest from all other events and idle, i.e., attackers discard any report of non-target events.
- **Multi-target:** attackers train multi-event classifiers to monitor all occurring events of a device, i.e., attackers accept reports of any event.

As for WiFinger, attackers use a consistent procedure to extract event fingerprints and apply (A)FMLCS to match them in all three scenarios.

C. Testbed Configurations & Dataset Collection

We built an automated event triggering system using a XiaoMi 8 mobile device to trigger IoT events. Traffic generated by the devices was sniffed and labeled accordingly. We tested the sniffing performance of a MacBook, a NetGear A6210 adapter [40], and an ALFA AWUS036ACH adapter [41], and finally chose the A6210 running on Ubuntu-16.04 for its best capture performance (lowest packet loss). For each IoT event, we generated 30 samples in a lab environment for training and 20 samples in a home environment for testing. All samples had approximately 40-second gaps between them to ensure completion of events. In total, our dataset includes 10 devices and 31 events (Table VI) representative of smart devices on the market, including smart home agents, small smart peripherals, and integrated smart actuators. Devices with simpler functionality are TP-Link Plug, Gosund Plug, ICX-RF Controller, and Wiz Hue Light. For complex devices, we chose Amazon Echo, Google Home, Xiaomi Smart Sweeper and humidifier, Midea Dishwasher and Dish Sterilizer. Apart from the tested devices, a laptop and a TV were connected to the same network serving as background noise traffic. All devices were connected via 2.4GHz Wi-Fi.

D. Results

1) Detection Performance: The results of the naive binary classification are shown in Table II, where three models perform similarly. Under the naive setting, Peek-a-boo obtains the highest recall rate of 96% and WiFinger achieves the highest precision rate of 98% on average.

Methods	Peek-a-boo RF		IoTBeholder		WiFinger(ours)	
	Rec.	Prec.	Rec.	Prec.	Rec.	Prec.
Average	**0.96**	0.78	0.92	0.88	0.90	**0.98**

TABLE II: Three models demonstrate similar performance on the naive scenario.

Advancing to the single-target setting, WiFinger remains the excellent performance, but Peek-a-boo and IoTBeholder start demonstrating a decreasing trend. As shown in Table III, Peek-a-boo achieves 83% recall and 81% precision, while IoTBeholder achieves 74% recall and 87% precision. The performance gap of between both models and WiFinger mainly lies in the precision. Due to the noisiness of WiFi traffic, flow-level features within a short window period vary significantly from time to time. This increases the difficulty of distinguishing events especially when their traffic fingerprints are similar.

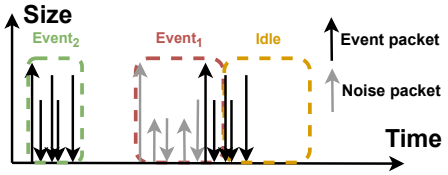


Fig. 13: The influence of mismatch in continuous event tracking. The model skips half of the $Event_2$ packets for misclassifying $Event_1$.

For example, E7 and E8 (also E9 and E10) only have byte-level differences inside a window, causing obvious degradation on Peek-a-boo and IoTBeholder.

In the most advanced and practical scenario, the performance gap between WiFinger and the other two models becomes very significant. WiFinger remains the highest recall and precision rates of 86% and 95%, while Peek-a-boo and IoTBeholder now only achieve 49% and 46% recall rates, and 48% and 35% precision rates respectively, barely useable.[3] Such results are reflecting the models' authentic performances on real-world tracking that have been overrated. Most of the existing works evaluate classification performances on chunked samples, where the results of different samples are classified independently. Nonetheless, in the multi-target scenario, misclassifications may impact the detections of subsequent events, as shown in Figure 13. Therefore, during event tracking, false positive detections have a much higher degradation on the actual overall performance, emphasizing the importance of high precision. As a comparison, WiFinger outstands for its excellent precision rate. Due to the event-unique base fingerprint features, one event hardly gets matched to a different fingerprint. Furthermore, we experimented matching fingerprints across devices, and their uniqueness demonstrate the capability of facilitating simultaneous device fingerprinting and event identification. Such cross-device uniqueness has also been verified in previous works [1], [5]. In later sections, we further provide more detailed packet-level analysis for the mismatches of WiFinger.

2) Event Base Fingerprint: Case Study: We demonstrate the metadata of the extracted base fingerprints in Table IV. First, complex devices may have very different fingerprint lengths and durations, depending on the commands' complexity. Alexa Echo's fingerprints' (E1-E6) lengths vary from 4 to 41 packets, and the duration varies from 0.2s to 7.7s. Second, correlated events mostly have very similar pattern. For instance, events in E11 (or E12-E15) have exactly the same fingerprints, i.e., same fingerprint lengths, packet sizes, directions, and similar interval distributions. These events are inseparable from encrypted traffic analysis. Nevertheless, some correlated events share subtle differences: on/off commands

[3]Considering that 30 samples may not be sufficient for training robust ML-based models for multi-event classification tasks, we tried to further collect 100 training samples for each event, but observe no performance improvement.

Event ID	Single-target				Multi-target				Single-target/Multi-target	
	Peek-a-boo RF		IoTBeholder		Peek-a-boo RF		IoTBeholder		WiFinger(ours)	
	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision
E1	0.9	1	0.2	0.67	0.97	1	0.15	0.3	0.9	0.72
E2	1	0.9	1	1	0.05	0.04	0.25	0.25	1	1
E3	0.95	1	0.95	1	0.15	0.11	0.1	0.07	0.95	1
E4	1	0.8	1	1	0.1	0.08	0.1	0.06	1	1
E5	0.7	0.47	1	0.69	0.15	0.08	0.5	0.29	0.95	1
E6	0.9	1	0.05	1	0.97	1	0.05	0.08	0.95	1
E7	0.95	0.41	1	0.74	0.7	0.44	1	0.51	0.7	1
E8	0.95	0.42	1	0.69	0.9	0.51	1	0.5	0.8	1
E9	0.75	0.94	0.9	0.95	0.65	0.41	0.9	0.47	0.85	0.77
E10	0.9	0.72	1	0.59	0.8	0.46	1	0.51	0.75	0.88
E14	0.9	0.86	0.53	1	0.55	0.58	0.2	0.12	0.85	1
E15	0.95	0.97	0.83	0.94	0.85	0.63	0.7	0.93	0.65	1
E18	0.2	1	0.38	1	0.8	0.97	0.05	0.2	0.68	1
E19	0.85	0.97	0.6	1	0	0	0.23	0.43	0.58	1
Average	0.83	0.81	0.74	0.87	0.49	0.48	0.46	0.35	**0.85**	**0.95**

TABLE III: Event detection performance under the single-target and multi-target scenarios. The results of WiFinger under two scenarios are completely and thus merged.

of the Wiz Hue Light (E9-E10) have single-byte differences on their first two fingerprint packets; the volume up/down commands of the Google Home (E7-E8) also have single-byte differences on several fingerprint packets, but the two commands also have different lengths as well. Last but not least, complex events traffic typically involve more sub-bursts (segments) during their span. These sub-bursts increase the difficulty of burst-classification or window-selection for ML-based approaches, but turn out to be useful for handling large volume of traffic with our segmentation technique. In general, IoT fingerprints have both significant flow-level differences and subtle packet-level variations, making it harder for tuning ML models to capture their traffic patterns.

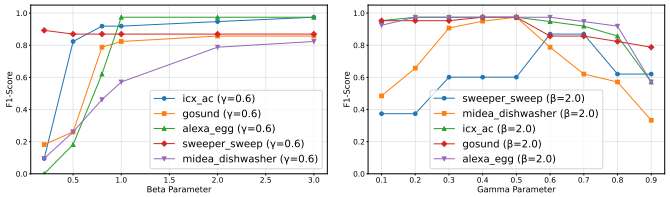
Event ID	Packet Num	Duration(s)	Sub-bursts
E1	4	0.23	1
E2	23	4.5	3
E3	41	7.7	6
E4	28	3.15	3
E5	42	4.03	4
E6	16	0.35	1
E7	9	3.29	2
E8	17	3.41	3
E9	4	0.5	1
E10	4	0.5	1
E11	4	0.2	1
E12	2	0.04	1
E13	20	1.52	2
E14	8	5.69	2
E15	12	5.76	2

TABLE IV: Metadata of the extracted base fingerprints.

3) Fingerprint Robustness: Noise Sensitivity: We manually analyze the matching results of FNs and FPs to identify their main causes. **FN:** WiFinger drops a detection result either for not matching enough packets of the base fingerprint or the failure of aligning subsequences temporally, where the former situation happens much more frequently. In our experiment settings, a successful match must comprise 60% of the base fingerprint (except Mi Sweeper). However, when the base fingerprints are short as 2-4 packets, any packet loss has significant impact on the detection results. For example, E12 of Mi Sweeper has a fingerprint involving only 2 packets, indicating that any missing packet will directly causing the failure of detection. **FP:** we also notice that packet loss is the main reason for misclassifying events with similar base fingerprints. For example, E7 and E8 adjust the volume of the Google Home speaker, and a few fingerprint packets of E7 are a-byte larger than E8. Nevertheless, if such characteristic distinguishing packets are lost from E8, its remaining packets could be perfectly mismatched as an E7 event. Such mismatches also happen to some idle states whose traffic exhibits similar patterns. For instance, idle traffic of Mi Sweeper can occasionally match 50% or 75% of the E11 fingerprint. Diving deeper into the situation, we notice that such special phenomena do not happen to any other devices/events and only occurs after the "stop sweeping" commands. A reasonable explanation is that there exists hidden events happening to share similar traffic patterns, e.g., reporting the "back-to-charging" status.

In summary, short base fingerprints are much more sensitive to packet losses, causing the majority of the FNs and FPs. Nonetheless, depending on the needs, attackers can adjust the matching percentage parameter for these events to balance the trade-off between precision and recall. If attackers increase the matching percentage γ, they could obtain higher precision but lower recall, or vice versa. For example, during the experiment, we increase γ for Mi Sweeper device to 100% to ensure that no FP result is detected, at the cost of degrading the recall rate to 65% and 85%.

4) Ablation Study on Parameter Setting: While attackers could adapt parameters to their need on precision and recall rates, understanding a balanced setting is also crucial. WiFinger has three parameters that could be adjusted: similar packet size gap ϵ, minimum sequence matching percentage γ, and interval distance threshold β. During the experiment, we empirically set ϵ as 1, meaning two packets are only considered similar if they have the exact same sizes and



(a) F1-score v.s. β (b) F1-score v.s. γ

Fig. 14: Ablation study on parameter β and γ.

Event ID	Packet Num	Process Time(s)		F1 Score	
		FMLCS	AFMLCS	FMLCS	AFMLCS
E1	3854	0.13	0.13	0.85	0.80
E2	5740	0.17	0.07	1.00	1.00
E3	8178	2.34	0.15	0.92	0.97
E4	12670	7.97	0.85	0.97	1.00
E5	18011	39.2	1.3	0.33	0.97
E6	1789	0.14	0.15	1.00	0.97
E7/E8	4179	0.34	0.1	0.84	0.86

TABLE V: AFMLCS improves both efficiency and accuracy for large-volume traffic events.

transmission directions. The influence of β and γ on the performance is shown in Figure 14.

By increasing β, WiFinger accommodates more network jitters during the transmission and reaches a stable performance when β is larger than 2 seconds. Generalizing to various network conditions, setting $\beta = 3$ is sufficiently accommodating. Additionally, WiFinger achieves best performance when γ is around 0.5-0.6. There are several factors influencing the value. First, higher γ suggests higher similarity to the base fingerprint in terms of packet sizes and orders, making the results more precise, and vice versa. Therefore, either too high or low γ results in the increment of false negatives or false positives, respectively, lowering the F1-score. Second, γ also relates to the sniffing packet loss rate. In an extreme case where all packets could be sniffed, attackers would expect matching the whole base fingerprint all the time, i.e., $\gamma = 1$. According to our experiments (under a packet loss rate of around 20%), we suggest initially set γ as 0.5-0.6, and increase it according to actual sniffing capability. Lower γ may work, but only for very unique and long fingerprints.

5) AFMLCS v.s. FMLCS: We introduced anchor reference and fingerprint segmentation to optimize the efficiency and accuracy of FMLCS for large-volume traffic devices (E1-E8). We use the multi-target setting to compare their online matching performance. Each testing file contains 20 events spanning over 1200 seconds. As shown in Table V, AFMLCS requires significantly less time to process the traffic, and the time cost difference increases with traffic volume. For example, during a sudden burst of traffic exceeding one hundred seconds in Event 5 (E5), FMLCS spends most of the time handling the burst but exhibits even worse performance.

Moreover, AFMLCS also plays a very important role in the base fingerprint extraction. During the extraction, WiFinger applies FMLCS to extract consensus sequences pairwisely among all event traffic bursts. However, unlike the online matching paradigm that matches a noisy target sequence to a clean base fingerprint, offline extraction discovers matches between two noisy sequences involving a significant amount of similar packets. As a result, baseline FMLCS got stuck at the DP function for its exponential time complexity. During the experiments, FMLCS could not finish the process for E2-E8 even after hours of waiting, while AFMLCS only takes less than a minute to do so. In general, for short fingerprints/traffic where segmentation techniques are not applicable, WiFinger degrades AFMLCS to the baseline version for its simplicity and effectiveness.

E. Countermeasures

To protect users from such privacy inference attacks, previous works have proposed several countermeasures, including traffic shaping [23], traffic delaying [42], and packet padding [43], [44]. Traffic delaying delays packet transmission for a random short period of time to obfuscate the interval patterns between packets. Traffic shaping randomly insert dummy packets in both direction to change the flow-level patterns and hide flow bursting characteristics. Packet padding adds dummy bytes to each packet to obfuscate the size of payload. To emulate these protection mechanisms, we add random delays (0-0.05s or 0-0.2s) to packets for traffic delaying, add dummy packets to the original traffic for traffic shaping, and slightly increase all packet sizes (1-5 bytes randomly) for packing padding defense, respectively. We evaluate current fingerprints against the three defenses under the multi-target setting, and the effectiveness of these countermeasures against WiFinger is shown in Figure 15. Generally, WiFinger is more robust to various defenses than RF and IoTBeholder, which are mostly disabled by the three defenses.

First, traffic shaping has very limited impact on the WiFinger's performance. This is because dummy packets does not influence the intervals between fingerprint packets as well as their sizes and directions. Consequently, the shaped traffic still embodies the same event patterns, and thus most events end up with a similar performance with a slight drop of the recall rate. Nonetheless, due to the significant amount of packets being added, WiFinger spent more time to process the traffic and may fail to finish the process for complicated events timely.

Traffic delaying is more effective due to its variations on the intervals between fingerprint packets. During the defense, transmission delays will be accumulated for each packet, i.e., if a request is delayed, its response will be also delayed for the same amount of time plus its own random delay. Consequently, as the volume of packets being transmitted increases, the intervals between tail packets and header packets increases significantly. When such variations exceeds the temporal alignment threshold of FMLCS, events cannot be detected effectively. Nonetheless, we notice that traffic delaying has biased performances among events. For instance, it has high impact on long-fingerprint events with large traffic volume, dropping the recall/precision rates for E2-E5 to 0%. Yet, when

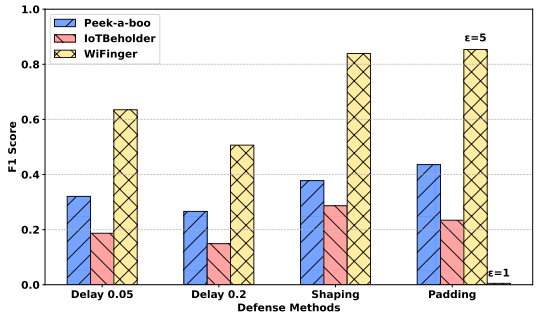


Fig. 15: The three methods' attacking performances against different defenses.

the event traffic is short, accumulated delay of 0.1-0.2s has bare effect on WiFinger's performance.

Traffic padding demonstrates to be the most effective defense against WiFinger if no further adaptation is applied. AFMLCS matches packets by examining their sizes and directions. Due to the direct change of the packet sizes, the whole matching process is significantly deconstructed. As a result, the recall and precision rates drop to 0% for all events if WiFinger still requires exact matches on packet sizes, i.e., $\epsilon = 1$. However, as we adapt WiFinger to the defense by loosing ϵ to 5 bytes, the performance of WiFinger recovers to a degree as if no defense is applied. For a padding defense to be effective, we suggest that it should add enough dummy bytes (e.g., ≥ 100) so that packets of different types could be padded to a similar size level and get obfuscated.

Cost-Effectiveness tradeoff: Since WiFinger aims at the first-hand WiFi traffic, protections must be implemented on the devices to influence the traffic patterns sniffed by attackers. As a result, this inevitably increases the burden of IoT application developers and manufactures. Moreover, the three defenses come at different costs.[4]

Traffic shaping takes more bandwidth for the significant amount of extra dummy packets to be transmitted. Transmission rates of events are several times (from two to over ten times) higher than their regular rate, causing unacceptable overhead. **Traffic delaying** increases the latency of an event execution. For large-volume traffic events, their full executions are extended 7 to 30 seconds, severely impacting the device utility. Although the accumulated delays of 0.5-1 seconds for small-volume traffic event maintain the regular functioning, they are not sufficient for protecting traffic from WiFinger attacks. **Packet padding** applies relative small packet-level overheads, but achieves strong effects on breaking fingerprinting approaches relying on the size of the payload. By randomly padding 1-100 dummy bytes to packets, it disables WiFinger at the cost of around 10% overhead on the transmission bandwidth. Therefore, we suggest that packet padding should be the first line of defenses against WiFinger, as well as other similar fingerprinting attacks. Some valid implementation options of randomizing the payload sizes include using OkHttp [45] to pad HTTP headers or using padding functionalities of the middleware protocols such as TLS. The results also indicate a urgent need for actual implementation of random padding defenses in real IoT systems.

[4]Detailed defense costs can be found in Appendix C

VI. Discussion

Streaming IoT Devices/Events. Streaming IoT devices such as Smart Cameras and Smart Doorbells belong to another dominant class. When turned on, these devices constantly stream video data to the cloud using the UDP protocol, and users can always view such data lively via the companion app. However, when triggering commands such as "photo capture" or "video recording", some experimented devices (e.g., Wyze Camera, XiaoMi Camera) did not demonstrate characteristic traffic patterns. Diving deeper into their TCP/IP traffic [46], we notice that no TCP packet was transmitted during events, yet photos and videos are always successfully stored locally on the mobile device. This indicates that "photo capture" and "video recording" for some cameras are actually local operations where the mobile device takes a snapshot of the video stream. Therefore, for streaming devices, some events are not observable from the network traffic perspective and cannot be fingerprinted at all. Nonetheless, most streaming devices demonstrate obvious traffic volume differences that can be utilized to determine their on/off states or "motion detected" events [7].

Potentials Beyond WiFi and Smart Home. WiFinger has the potential to be applied to other non-invasive benign monitoring scenarios. For example, in the healthcare domain, fingerprinting medical devices such as blood glucose meters and heart rate monitors provides multi-modal information for detecting anomaly behaviors [47], e.g., uneven heartbeat. In the industrial domain, WiFinger can be used to fingerprint smart meters or smart sensors and help detect anomalies in the power grid or manufacturing processes. Apart from IoT systems, WiFinger also demonstrates its potential of fingerprinting complex network events. For instance, fingerprinting mobile application behaviors can be useful for unveiling privacy leaking activities in the background by detecting known network traffic patterns corresponding to different apps [48], [49].

VII. Conclusion

In this work, we proposed WiFinger, an explainable and granular packet-level IoT event fingerprinting approach based on a sequence-matching algorithm. We demonstrated that the current trend of using ML approaches for fingerprinting IoT events has inherent overheads and limitations, especially when applied to noisy Wi-Fi traffic. Additionally, we identified a gap in existing evaluation methodologies, which often use chunked traffic samples rather than the more appropriate approach of detecting events within continuous traffic streams. Our experiments show that WiFinger achieves the best performance under more practical settings, maintaining very low false positive rates.

REFERENCES

[1] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, "Ping-pong: Packet-level signatures for smart home device events," *arXiv preprint arXiv:1907.11797*, 2019.

[2] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer iot devices: A multi-dimensional, network-informed measurement approach," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 267–279.

[3] B. Copos, K. Levitt, M. Bishop, and J. Rowe, "Is anybody home? inferring activity from smart home network traffic," in *2016 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2016, pp. 245–251.

[4] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *2019 IEEE symposium on security and privacy (sp)*. IEEE, 2019, pp. 1362–1380.

[5] Y. Wan, K. Xu, F. Wang, and G. Xue, "Iotathena: Unveiling iot device activities from network traffic," *IEEE Transactions on Wireless Communications*, vol. 21, no. 1, pp. 651–664, 2021.

[6] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi, "Homesnitch: Behavior transparency and control for smart home iot devices," in *Proceedings of the 12th conference on security and privacy in wireless and mobile networks*, 2019, pp. 128–138.

[7] N. Apthorpe, D. Reisman, S. Sundaresan, A. Narayanan, and N. Feamster, "Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic," *arXiv preprint arXiv:1708.05044*, 2017.

[8] A. Acar, H. Fereidooni, T. Abera, A. K. Sikder, M. Miettinen, H. Aksu, M. Conti, A.-R. Sadeghi, and S. Uluagac, "Peek-a-boo: I see your smart home activities, even encrypted!" in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 207–218.

[9] Q. Zou, Q. Li, R. Li, Y. Huang, G. Tyson, J. Xiao, and Y. Jiang, "Iotbeholder: A privacy snooping attack on user habitual behaviors from smart home wi-fi traffic," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 7, no. 1, pp. 1–26, 2023.

[10] J. Li, H. Zhou, S. Wu, X. Luo, T. Wang, X. Zhan, and X. Ma, "{FOAP}:{Fine-Grained}{Open-World} android app fingerprinting," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1579–1596.

[11] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "Mimetic: Mobile encrypted traffic classification using multimodal deep learning," *Computer networks*, vol. 165, p. 106944, 2019.

[12] 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 Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[13] S. Rezaei, B. Kroencke, and X. Liu, "Large-scale mobile app identification using deep learning," *IEEE Access*, vol. 8, pp. 348–362, 2019.

[14] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying iot devices in smart environments using network traffic characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, 2018.

[15] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 2017, pp. 2177–2184.

[16] S. Dong, Z. Li, D. Tang, J. Chen, M. Sun, and K. Zhang, "Your smart home can't keep a secret: Towards automated fingerprinting of iot traffic," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, 2020, pp. 47–59.

[17] X. Ma, J. Qu, J. Li, J. C. Lui, Z. Li, W. Liu, and X. Guan, "Inferring hidden iot devices and user interactions via spatial-temporal traffic fingerprinting," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 394–408, 2021.

[18] M. Alyami, I. Alharbi, C. Zou, Y. Solihin, and K. Ackerman, "Wifi-based iot devices profiling attack based on eavesdropping of encrypted wifi traffic," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2022, pp. 385–392.

[19] S. J. Saidi, A. M. Mandalari, R. Kolcun, H. Haddadi, D. J. Dubois, D. Choffnes, G. Smaragdakis, and A. Feldmann, "A haystack full of needles: Scalable detection of iot devices in the wild," in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 87–100.

[20] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New directions in automated traffic analysis," in *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, 2021, pp. 3366–3383.

[21] I. Analytics, "State of iot 2024: Number of connected iot devices growing 13

[22] W. Alliance, "Discovery wifi: Security." [Online]. Available: https://www.wi-fi.org/discover-wi-fi/security

[23] N. Apthorpe, D. Y. Huang, D. Reisman, A. Narayanan, and N. Feamster, "Keeping the smart home private with smart (er) iot traffic shaping," *arXiv preprint arXiv:1812.00955*, 2018.

[24] D. Y. Huang, N. Apthorpe, F. Li, G. Acar, and N. Feamster, "Iot inspector: Crowdsourcing labeled network traffic from smart home devices at scale," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 2, pp. 1–21, 2020.

[25] G. Wan, S. Liu, F. Bronzino, N. Feamster, and Z. Durumeric, "{CATO}:{End-to-End} optimization of {ML-Based} traffic analysis pipelines," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, 2025, pp. 1523–1540.

[26] K. Fauvel, F. Chen, and D. Rossi, "A Lightweight, Efficient and Explainable-by-Design Convolutional Neural Network for Internet Traffic Classification," in *Proceedings of the 29th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2023.

[27] Y. Jin, E. Sharafuddin, and Z.-L. Zhang, "Unveiling core network-wide communication patterns through application traffic activity graph decomposition," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1, pp. 49–60, 2009.

[28] S. Zhao, M. Chandrashekar, Y. Lee, and D. Medhi, "Real-time network anomaly detection system using machine learning," in *2015 11th international conference on the design of reliable communication networks (drcn)*. IEEE, 2015, pp. 267–270.

[29] M. K. Hooshmand, M. D. Huchaiah, A. R. Alzighaibi, H. Hashim, E.-S. Atlam, and I. Gad, "Robust network anomaly detection using ensemble learning approach and explainable artificial intelligence (xai)," *Alexandria Engineering Journal*, vol. 94, pp. 120–130, 2024.

[30] S. Naseer, Y. Saleem, S. Khalid, M. K. Bashir, J. Han, M. M. Iqbal, and K. Han, "Enhanced network anomaly detection based on deep neural networks," *IEEE access*, vol. 6, pp. 48 231–48 246, 2018.

[31] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning ddos detection for consumer internet of things devices," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 29–35.

[32] M. Idhammad, K. Afdel, and M. Belouch, "Semi-supervised machine learning approach for ddos detection," *Applied Intelligence*, vol. 48, pp. 3193–3208, 2018.

[33] S. Nanda, F. Zafari, C. DeCusatis, E. Wedaa, and B. Yang, "Predicting network attack patterns in sdn using machine learning approach," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2016, pp. 167–172.

[34] A. Churcher, R. Ullah, J. Ahmad, S. Ur Rehman, F. Masood, M. Gogate, F. Alqahtani, B. Nour, and W. J. Buchanan, "An experimental analysis of attack classification using machine learning in iot networks," *Sensors*, vol. 21, no. 2, p. 446, 2021.

[35] T. Gu, Z. Fang, A. Abhishek, H. Fu, P. Hu, and P. Mohapatra, "Iotgaze: Iot security enforcement via wireless context analysis," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 884–893.

[36] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: Monitoring smart home apps from encrypted traffic," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1074–1088.

[37] R. Perdisci, T. Papastergiou, O. Alrawi, and M. Antonakakis, "Iotfinder: Efficient large-scale identification of iot devices via passive dns traffic analysis," in *2020 IEEE european symposium on security and privacy (EuroS&P)*. IEEE, 2020, pp. 474–489.

[38] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola, "Autogluon-tabular: Robust and accurate automl for structured data," *arXiv preprint arXiv:2003.06505*, 2020.

[39] A. Developers, "Android debug bridge." [Online]. Available: https://developer.android.com/tools/adb

[40] Netgear, "Netgear a6210." [Online]. Available: https://www.netgear.com/home/wifi/adapters/a6210/

[41] ALFA, "Alfa awus036ach." [Online]. Available: https://www.alfa.com.tw/products/awus036ach_1

[42] X. Cai, R. Nithyanand, and R. Johnson, "Cs-buflo: A congestion sensitive website fingerprinting defense," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, 2014, pp. 121–130.
[43] 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 *2012 IEEE symposium on security and privacy*. IEEE, 2012, pp. 332–346.
[44] X. Cai, R. Nithyanand, T. Wang, R. Johnson, and I. Goldberg, "A systematic approach to developing and evaluating website fingerprinting defenses," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 227–238.
[45] OkHttp, "Okhttp webpage." [Online]. Available: https://square.github.io/okhttp/
[46] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information Exposure for Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach," in *Proc. of the Internet Measurement Conference (IMC)*, 2019.
[47] N. Mashnoor and B. Charyyev, "Network traffic analysis of medical devices," in *2024 International Conference on Smart Applications, Communications and Networking (SmartNets)*. IEEE, 2024, pp. 1–6.
[48] T. Van Ede, R. Bortolameotti, A. Continella, J. Ren, D. J. Dubois, M. Lindorfer, D. Choffnes, M. Van Steen, and A. Peter, "Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic," in *Network and distributed system security symposium (NDSS)*, vol. 27, 2020.
[49] M. Jiang, Z. Li, P. Fu, W. Cai, M. Cui, G. Xiong, and G. Gou, "Accurate mobile-app fingerprinting using flow-level relationship with graph neural networks," *Computer Networks*, vol. 217, p. 109309, 2022.

APPENDIX

A. NP-hard Proof

To prove the NP-hardness of the problem, we reduce the *Maximum Cardinality Subset* (MCS) problem to the *FMLCS with time constraints* problem. The *Maximum Cardinality Subset* problem is defined as follows. Giving a non negative set,

$$S = \{a_1, a_2, a_3, \ldots, a_n\}$$

the target is to find a subset $S' \subseteq S$ with the most elements such that the sum of the elements in S' does not exceed threshold K. The *Maximum Cardinality Subset* problem has already been proven to be NP-complete.

1) Reduction: We reduce the MCS problem to the FMLCS problem. For the FMLCS problem, two sequences are constructed:

$$\begin{aligned} Seq_1 = [&\{time : \sqrt{a_1}, size : 1, dir : 1\}, \\ &\{time : \sqrt{a_2}, size : 1, dir : 1\}, \\ &\{time : \sqrt{a_3}, size : 1, dir : 1\}, \ldots, \\ &\{time : \sqrt{a_n}, size : 1, dir : 1\}], \\ a_i \in S& \end{aligned} \tag{1}$$

$$\begin{aligned} Seq_2 = [&\{time : 0, size : 1, dir : 1\}, \\ &\{time : 0, size : 1, dir : 1\}, \\ &\{time : 0, size : 1, dir : 1\}, \ldots, \\ &\{time : 0, size : 1, dir : 1\}] \end{aligned} \tag{2}$$

For each element $a_i \in S$, we construct a sequence Seq_1 with each element using a_i as the timestamp, and another sequence Seq_2 with all elements having $time = 0$. As the elements in both Seq have the same size, the only constraint is that the L2Norm distance between timestamps of the selected subsequence should be lower than $\sqrt{K}$.

2) Equivalence: If there exists a subset $S' \subseteq S$ such that the sum of the elements in S' does not exceed threshold K, then the FMLCS problem has a solution. The solution is to select the elements in Seq_1 that correspond to the elements in S', and the L2Norm distance between the timestamps of the selected elements is less than $\sqrt{K}$.

$$\sum_{a_i \in S'} a_i \leq K \Rightarrow \sum_{a_i \in S'} (\sqrt{a_i} - 0)^2 \leq (\sqrt{K})^2 \tag{3}$$

Therefore, since MCS is NP-Complete, FMLCS has a polynomial time solution only if P=NP.

B. Events ID

Table VI lists the events and devices used in the experiments.

C. Detailed Defense Overheads

Detailed event-level performance against the three defenses (delay0.2s, shaping, and padding) are shown in Fig. 16, Fig. 17, and Fig. 18. Results are expected to be close to the origin point (strong impact on performance with low costs). Randomly padding 1 - 100 bytes to packets introduce the lowest overhead to the bandwidth and complete prohibit WiFinger from identifying events.

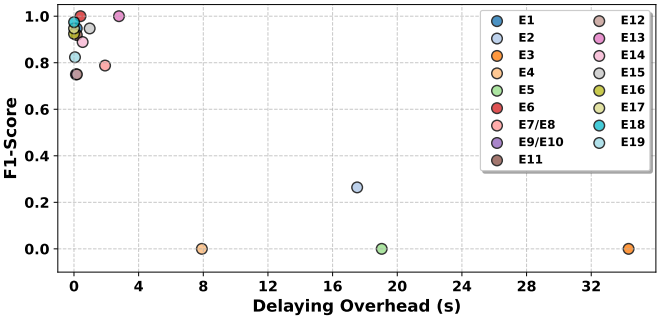


Fig. 16: Additional event execution overhead (seconds) introduced by traffic delaying defense.

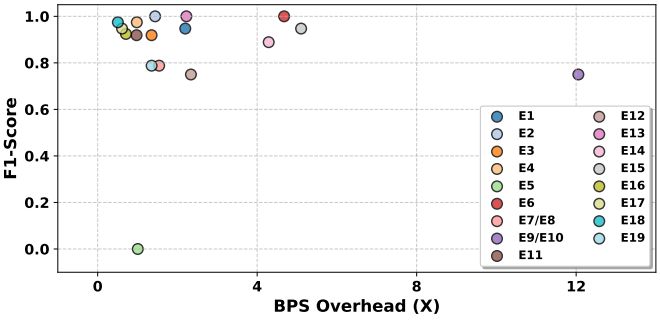


Fig. 17: Additional bandwidth overhead (# times of bps) introduced by traffic shaping defense.

ID	Device Type	Device	Related Events
E1	Agentic Controller	Alexa Echo Dot	DND/UnDND
E2 (H)			Q1: What time is it?
E3 (H)			Q2: What's the price of eggs?
E4 (H)			Q3: What's the weather now?
E5 (H)			Q4: What's the weather like in X?
E6			Volume Up/Down
E7 (H)		Google Home	Volume Up
E8 (H)			Volume Down
E9	Smart Peripherals	Wiz Hue Light	On
E10			Off
E11		TP-Link Plug	On/Off
E12		ICX-RF A/C Controller	On/Off
E13		Gosund Plug	On/Off
E14	Integrated Smart Actuator	Mi Sweeper	On/Off
E15			Mode Silent/Standard/Strong
E16		Midea Dishwasher	On/Off
E17		Midea Dish Sterilizer	On/Off
E18		Xiaomi Humidifier	On/Off
E19			Continuous humidification/Close

TABLE VI: Event and the corresponding IDs. "H" indicates high speed traffic events.

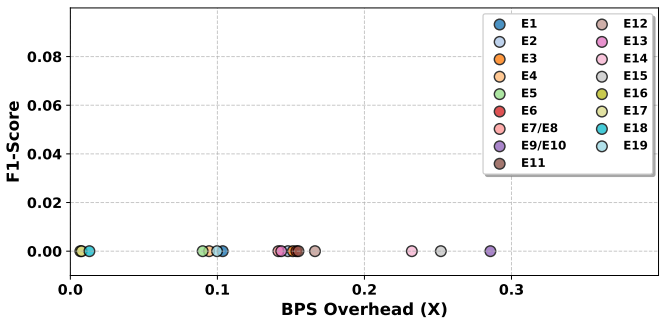


Fig. 18: Additional bandwidth overhead (# times of bps) introduced by packet padding defense.