Janus Varmarken*, Jad Al Aaraj, Rahmadi Trimananda, and Athina Markopoulou

FingerprinTV: Fingerprinting Smart TV Apps

Abstract: This paper proposes FingerprinTV, a fully automated methodology for extracting fingerprints from the network traffic of smart TV apps and assessing their performance. FingerprinTV (1) installs, repeatedly launches, and collects network traffic from smart TV apps; (2) extracts three different types of network fingerprints for each app, i.e., domain-based fingerprints (DBF), packet-pair-based fingerprints (PBF), and TLS-based fingerprints (TBF); and (3) analyzes the extracted fingerprints in terms of their prevalence, distinctiveness, and sizes. From applying FingerprinTV to the top-1000 apps of the three most popular smart TV platforms, we find that smart TV app network fingerprinting is feasible and effective: even the least prevalent type of fingerprint manifests itself in at least 68% of apps of each platform, and up to 89% of fingerprints uniquely identify a specific app when two fingerprinting techniques are used together. By analyzing apps that exhibit identical fingerprints, we find that these apps often stem from the same developer or "no code" app generation toolkit. Furthermore, we show that many apps that are present on all three platforms exhibit platform-specific fingerprints.

Keywords: Smart TV, connected TV, network measurement, privacy, fingerprinting

DOI 10.56553/popets-2022-0088
Received 2021-11-30; revised 2022-03-15; accepted 2022-03-16.

1 Introduction

A *smart TV* (or *connected TV*) is an Internet-connected TV with computational capabilities. These enhancements to the traditional TV set enables the smart TV to stream content from the Internet and run interactive applications. Smart TVs come in two flavors: (1) built-in smart TVs, where the hardware and software is integrated in a traditional TV set; and (2) over-the-top streaming devices, which are external dongles or boxes that transform a TV into a smart TV. In this paper, we use "smart TV" to refer to both flavors.

Over the past decade, consumers have increasingly adopted smart TV devices: between 2011 and 2021, the percentage of U.S. TV households with at least one smart TV has increased from 30% to 82% [1], and smart TV sets now outnumber traditional TV sets in American homes [2]. Smart TVs offer convenient access to streaming services such as Netflix and Hulu, a rich selection of games, and are also increasingly used as a central hub for entertainment in general, such as music streaming and social media [3]. While appealing from an entertainment standpoint, smart TVs also introduce new privacy risks. For example, researchers have shown that tracking is pervasive on smart TVs [4, 5]—even if the user opts to limit ad tracking [4]—and blocklists only provide limited protection against exposure of personally identifiable information [4, 5].

Motivation. In this paper, we set out to further characterize the privacy exposures of smart TVs by studying if a passive, in-network observer can identify what application ("app") is in use on a smart TV from the network traffic it generates, even if the traffic is encrypted. This type of problem is commonly referred to as *network fingerprinting* and has been studied extensively in the context of websites [6–24], desktop and mobile applications [25–28], and Internet of Things (IoT) devices with narrow functionality, such as smart light bulbs and smart plugs [29–33]. However, to the best of our knowledge, no work has explored the feasibility of smart TV app fingerprinting at scale. This constitutes a significant gap in the literature for the following reasons.

First, in the context of smartphones, app usage has been shown to be indicative of the user's demographics, personality, interests, preferences, and habits [34]. Assuming this carries over to smart TVs, and considering that viewing history is regarded a cornerstone of programmatic TV advertising [35], smart TV app usage data is arguably a valuable asset for businesses engaged in targeted advertising. Since Internet Service Providers (ISP) are known to collect and use rich information about their customers for advertising purposes [36], it

***Corresponding Author: Janus Varmarken:** University of California, Irvine, E-mail: jvarmark@uci.edu
Jad Al Aaraj: University of California, Irvine, E-mail: jalaaraj@uci.edu
Rahmadi Trimananda: University of California, Irvine, E-mail: rtrimana@uci.edu
Athina Markopoulou: University of California, Irvine, E-mail: athina@uci.edu

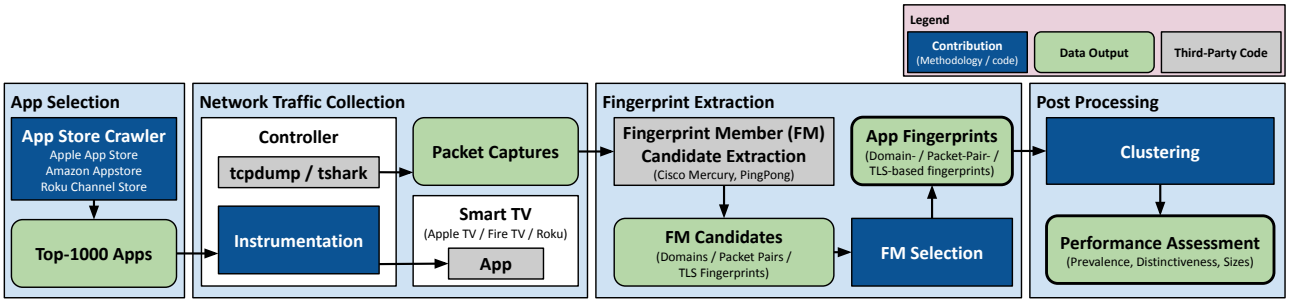

Fig. 1. Overview of FingerprinTV, a system for assessing the feasibility of fingerprinting smart TV apps. The Controller is a computer with both a wired and a wireless network interface, which is configured as a wireless access point with NAT. The smart TV is associated with this wireless network. FingerprinTV first crawls the app store of the smart TV platform to determine a list of apps to test (see Section 3.1). Next, FingerprinTV collects multiple samples of the "on-launch" traffic of each app in this list (see Section 3.2). FingerprinTV then processes the collected traffic samples to identify consistently occurring traffic, referred to as fingerprints (see Section 4.1). Finally, FingerprinTV assesses the resulting fingerprints' discriminative power using a methodology we devise that is based on agglomerative (hierarchical) clustering (see Sections 4.2 and 5).

is important to quantify to what extent they can track their customers' smart TV app usage as well.

Second, recognizing that television viewing history may reveal sensitive information about the viewer, such as religion and sexual orientation, the U.S. Congress has enacted laws that obligate (cable) companies to obtain consent from consumers before they collect and/or disclose viewing history, e.g., the Cable Privacy Act and the Video Privacy Protection Act [36]. Although the smart TV app in use may not reveal the exact content the user is watching in that respective app, it may still reveal the content's theme (e.g., religious, political, adult etc.) as many smart TV apps limit their offerings to a certain genre. Furthermore, for smart TV apps that offer access to a single live stream, the smart TV app in use is synonymous with the content being watched. Smart TV app fingerprinting may thus potentially constitute a violation of said laws.

Third, privacy concerns aside, smart TV app fingerprinting may have potential security implications, and can also be used for quality-of-service optimization. For example, an attacker who is aware of a vulnerability in a certain smart TV app can, by observing the manifestation of its fingerprint in live traffic, time when to launch their attack. Additionally, since most smart TV app traffic is bandwidth intensive, as it often involves video streaming, network operators may be interested in the ability to dynamically prioritize traffic from smart TVs when certain apps are in use.

Contributions. In order to empirically assess the feasibility of fingerprinting smart TV apps, we take the following steps.

First, we design and implement FingerprinTV, a fully automated system for assessing the feasibility and effectiveness of fingerprinting smart TV apps. An overview of FingerprinTV is provided in Figure 1. FingerprinTV (1) automatically installs, and repeatedly launches apps, while collecting their network traffic; (2) extracts a domain-based fingerprint (DBF), a packet-pair-based fingerprint (PBF), and a TLS-based fingerprint (TBF) from the network traffic of each app; and (3) assesses the extracted fingerprints' performance, in terms of their prevalence, distinctiveness, and sizes. To that end, we propose a methodology based on agglomerative clustering that (i) provides flexibility to make a trade-off between a fingerprint's size and its reliability, and (ii) is general enough to be applicable across all three types of fingerprints. We consider DBFs, PBFs, and TBFs because they are lightweight and only rely on a few packets per network flow, yet remain applicable even if an app's traffic is encrypted using TLS.

Second, we deploy FingerprinTV to collect network traffic from the top-1000 most reviewed smart TV apps of the three most widely used smart TV platforms, namely Apple TV, Fire TV, and Roku [37]. To the best of our knowledge, this is the first large-scale smart TV traffic dataset that also includes traffic from Apple TV apps. The dataset comprises 30K packet captures, 10K per platform.

Third, we analyze this dataset and provide the following findings and insights. We find that smart TV app fingerprinting is highly feasible and effective: even the least prevalent type of fingerprint manifests itself in at least 68% of apps of each platform. However, a fingerprint is only effective if it is distinct among other fingerprints. With this in mind, only DBFs and PBFs have merit, as up to 63% and 88% of the apps that exhibit DBFs and PBFs, respectively, have DBFs/PBFs that

are distinct among those of other apps of the same platform. We also find that if DBFs and PBFs are used in conjunction, 78%, 89%, and 76% of Apple TV, Fire TV, and Roku apps, respectively, have distinct fingerprints. Furthermore, our results show that when multiple apps exhibit an identical fingerprint, a common explanation is that these apps stem from the same developer, or have been generated using the same "no code" toolkit. Finally, we find that among 80 apps that are made available on all three smart TV platforms, 76% exhibit a *different* fingerprint on each platform, thus making it possible to not only fingerprint the smart TV app itself, but also to identify which platform it is being used on.

Network traffic features, such as destinations, packet sizes, and TLS configuration parameters, have been used to create fingerprints in other contexts [32, 38–46], but this paper is the first to consider them for smart TV apps. Our main contributions are the application and adaptation of existing families of techniques to smart TV apps, and a methodology that allows for automated extraction and evaluation of a range of fingerprints (i.e., DBFs, PBFs, and TBFs), in a uniform way and at scale. The implementation of our methodology can be used to repeat such assessments in the future, as smart TV apps and their fingerprints evolve. To that end, we plan to make the FingerprinTV code and dataset publicly available [47].

Outline. The remainder of this paper is structured as follows. Section 2 summarizes related work. Section 3 describes how we selected what apps to include in our study, and how FingerprinTV instruments the three smart TV platforms. Section 4 introduces the three fingerprinting techniques we consider, and explains the methodology we devise for assessing their performance. Section 5 presents the results for how well the three fingerprinting techniques perform. Section 6 compares fingerprints across platforms, and examines the benefits of using different fingerprinting techniques in conjunction. Section 7 discusses possible defenses, limitations, and future directions. Section 8 concludes the paper.

2 Background & Related Work

A network *fingerprint* (or *signature*) is network traffic that is characteristic for certain software (or hardware) and that may thus be used to identify the presence of such software (hardware) on the network. Fingerprinting has been studied by academics and professionals for decades, because it enables valuable services such as Network Intrusion Detection Systems and traffic prioritization schemes, yet at the same time also introduces privacy risks as it allows network operators to "spy" on individual users' computer usage.

Early fingerprinting techniques relied on applications' use of well known ports, but this approach had limited accuracy and granularity, which paved the way for proposals that relied on inspection of packet payload [48, 49]. Techniques based on payload inspection also proved applicable to recent emerging technologies, such as smartphones [50] and IoT [51], but their relevance is declining as the use of encryption is becoming more widespread in these technologies [52–54].

Fingerprinting Encrypted Traffic. With the introduction of SSL, the predecessor to TLS, researchers began exploring what information could (still) be inferred from network traffic, despite the payload being encrypted. For example, Bernaille and Teixeira [44] demonstrated that the application layer protocol used on top of SSL could be identified by analyzing the sizes and directions of the first few packets of an SSL session. Researchers have also reconstructed even more granular information from encrypted network traffic by fingerprinting HTTP User-Agent strings [55], websites [6, 7] (even in the presence of additional privacy enhancing technologies, such as encryption of domain names, tunneling, onion routing, and traffic morphing [8–24]), individual webpages on social media websites [56], desktop [25] and mobile [26–28] applications, and even individual user actions in mobile applications [57, 58] as well as voice commands on smart speakers [59–62] and individual functionality on IoT devices [29–33].

Some work has even shown that it is possible to fingerprint video content [63, 64]. Content fingerprinting provides more granular information than smart TV app fingerprinting, but is not as lightweight as the fingerprinting techniques we consider for smart TV apps. For example, Schuster et al. [64] employ Convolutional Neural Networks, and also note that data collection is a bottleneck as content must be played back in real-time, multiple times, and the technique proposed in [63] needs access to at least 30 Application Data Units, which equates to two minutes of video playback, before any inference attempts can be made.

Privacy of Smart TVs. While the literature on fingerprinting is evidently rich, to the best of our knowledge, no prior work has studied smart TV app fingerprinting at scale. A few papers have investigated fingerprinting in the context of smart TVs (alongside other IoT devices) [31, 32, 40, 51, 65]. However, these pa-

pers are concerned with fingerprinting the smart TV "as a whole", i.e., identifying its presence on the network [40, 51, 65], or with fingerprinting basic events on smart TVs, e.g., returning to the menu screen [31, 32], but do not attempt to fingerprint individual smart TV apps. Furthermore, since the network traffic profiles of smart TV apps differ from those of simpler smart home devices, existing fingerprinting techniques, such as PingPong [32], need to be modified accordingly. We discuss this in further detail in Section 4.1.2.

The work closest to ours is that of Moghaddam et al. [4] and our own prior work [5]. These papers were the first to study smart TV apps at scale, but focused on advertising and tracking on smart TVs. Our work continues the effort of understanding the privacy of smart TVs by focusing on fingerprinting smart TV apps. While we draw inspiration from [4, 5] w.r.t. how we instrument Fire TV and Roku, large-scale assessment of fingerprinting techniques on smart TV apps was not possible with prior work, and no dataset with multiple samples of apps' on-launch traffic previously existed. Additionally, whereas the instrumentation tools [66, 67] published alongside [5] only cover Fire TV and Roku, and still require manual intervention, FingerprinTV fully automates app testing, and adds support for Apple TV.

Other work that addresses the privacy of smart TVs includes: a qualitative assessment of the privacy of 10 popular streaming apps and five popular smart TVs [68]; a study that used crowd-sourcing to investigate what tracking domains smart TVs in real users' homes contact [54]; studies of the privacy risks of HbbTV (a technology for overlaying web content over regular TV channels), such as the broadcasters' ability to track users' viewing habits [69–71]; and two surveys [72, 73] of consumers' understanding of the privacy risks and practices of smart TVs that found clear evidence of widespread unawareness and confusion.

3 Data Collection

This section describes the design of FingerprinTV's data collection functionality (the App Selection and Network Traffic Collection boxes in Figure 1), and how we use FingerprinTV to collect network traffic from the top-1000 apps of each of the three most widely used smart TV platforms [37], namely Apple TV, Fire TV, and Roku. Section 3.1 explains how we use FingerprinTV to determine what apps to test. Section 3.2 explains how FingerprinTV automates interaction with the smart TVs. We summarize the dataset we collect using FingerprinTV in Section 3.3.

3.1 App Selection

To assess the feasibility of fingerprinting smart TV apps, we must test a large number of popular apps. To this end, we first use FingerprinTV to crawl the web interfaces of the app stores of the three smart TV platforms to obtain metadata for all available *free* apps of each platform. Drawing inspiration from [4, 5], we then use the number of user ratings submitted for an app as a gauge for the number of users of that app, and pick the 1000 free apps with the most ratings for each platform. We refer to this selection as the respective platform's top-1000 apps. Below, we briefly discuss the crawlers and, to add context to the top-1000s, report how many apps they discovered.

Apple TV. The Apple TV platform was not covered in [4, 5]. We implement our own crawler that traverses Apple's "iTunes Preview" website [74], which lists metadata for all apps (across all Apple platforms) that are made available on Apple's App Store. The crawl was performed in January 2021 and returned a total of 3,841 free Apple TV apps.

Fire TV. We also implement our own crawler for Fire TV since [5] does not provide a crawler, and since the crawler [75] provided alongside [4] does not log all app metadata necessary for our purposes. Our crawler determines the free apps that are compatible with our Fire TV model ("Fire TV Cube 2nd Generation"). This crawl was performed in March 2021 and returned a total of 6,870 free Fire TV apps.

Roku. For Roku, we use the scripts [67] provided alongside [5] to crawl the Roku Channel Store. The crawl was performed in May 2021 and returned a total of 14,246 free Roku apps.

3.2 Automation

To enable collection of apps' on-launch network traffic at scale, FingerprinTV instruments the Apple TV, Fire TV, and Roku platforms to repeatedly launch each app while collecting the smart TV's network traffic. In this section, we first provide a platform-agnostic overview of the hardware setup and the instrumentation procedure, followed by platform-specific implementation details.

Hardware Setup. The general setup for the instrumentation is depicted in the Network Traffic Collection

box of Figure 1. The Controller is a computer with a wired and a wireless network interface that runs a Unix-based operating system. It is configured as a wireless access point (with NAT), and the smart TV under test is associated with this wireless network. The Controller is also responsible for executing the instrumentation code and logging the network traffic from/to the smart TV.

Instrumentation Procedure. Each app is subjected to the same four-step instrumentation procedure:

1. The instrumentation first installs the app.
2. Next, it performs three "warm-up" launches of the app, without logging any network traffic. In each warm-up launch, the instrumentation emulates one of three sequences of key presses on the smart TV's physical remote. The purpose of these warm-up launches is to dismiss any terms of service and/or initial setup screens (e.g., for selecting viewing preferences) that only appear at first launch or until the user has made their choice(s); this way, we fingerprint how the app behaves during daily use, and not during first use. Drawing inspiration from [4], we pick the three most common key press sequences for dealing with terms of service and initial setup screens among the top-100 apps.
3. The instrumentation then enters its main phase where it collects L samples of the on-launch traffic for the app. For each sample, the instrumentation (1) kills the app to ensure it is not already running; (2) starts capturing traffic on the Controller's wireless interface; (3) launches the app; (4) waits for approximately 45 seconds; and (5) terminates the traffic capture. This produces L traffic captures (PCAP files), each of which contains all the traffic that occurred during a single launch of the app. We use the term *launch sample* to refer to such a capture.
4. Finally, the instrumentation uninstalls the app to free up space on the smart TV.

Apple TV Implementation Details. We use a MacBook Pro Retina (15 inch, Mid 2012), running macOS Catalina, as the Controller in Figure 1. We use an Apple Thunderbolt-to-Ethernet adapter to transform one of the MacBook's Thunderbolt ports into an Ethernet port, and connect this interface to the WAN. The MacBook's wireless interface is configured as a wireless access point, to which the Apple TV is connected. The instrumentation procedure is implemented using Apple's testing framework, XCUITest [76], (for controlling the Apple TV programmatically) and tshark (for capturing all network traffic passing through the wireless interface of the MacBook). We note that for Apple TV, the Controller must be a macOS-based system as the XCUITest APIs are only supported on macOS.

Fire TV Implementation Details. We use a Raspberry Pi 4 (8GB RAM) as the Controller in Figure 1. During data collection, the Raspberry Pi's onboard wireless radio proved unreliable, so we replaced it with a TP-Link Archer T3U Plus Wi-Fi adapter. The instrumentation is realized through a Python implementation [77] of the Android Debug Bridge [78] (for controlling the Fire TV programmatically) and tcpdump (for capturing all network traffic passing through the wireless interface of the Raspberry Pi).

Roku Implementation Details. We use a Raspberry Pi 4 (8GB RAM) with a TP-Link Archer T3U Plus Wi-Fi adapter, as the Controller in Figure 1. The instrumentation is realized using the Roku External Control Protocol [79] (for controlling the Roku device programmatically) and tcpdump (for capturing all network traffic at the wireless interface of the Raspberry Pi).

3.3 Dataset Summary

We deploy FingerprinTV and collect $L = 10$ samples of the network traffic generated at launch time by each of the 1000 most widely used apps on Apple TV, Fire TV, and Roku. The resulting dataset comprises 30K packet captures, 10K per platform.

4 Data Analysis

This section describes how FingerprinTV analyzes network traffic to assess the feasibility of fingerprinting smart TV apps. Section 4.1 introduces the three fingerprinting techniques we consider in this paper. Section 4.2 introduces the methodology that FingerprinTV uses to assess the performance of each fingerprinting technique in a uniform manner.

4.1 Fingerprinting Techniques

In the context of this paper, a *fingerprinting technique* is any algorithm that identifies what network traffic (if any) consistently occurs whenever a specific smart TV app is launched. In other words, a fingerprinting technique *extracts* fingerprints from a training network traffic dataset, and the extracted fingerprints can then later

be used to identify the corresponding apps in live traffic. This section introduces the three fingerprinting techniques we consider in this paper and implement support for in FingerprinTV (each fingerprinting technique is responsible for carrying out the steps in the Fingerprint Extraction box of Figure 1). We pick these particular techniques because they are lightweight, as they only rely on a few packets per flow, yet applicable even if an app's communication is encrypted using TLS. While the research community has explored these fingerprinting techniques in other contexts [32, 38–46], our work is the first to apply them to smart TV apps at scale.

4.1.1 Domain-Based Fingerprints (DBF)

The first fingerprinting technique we consider identifies an app based on the set of domains the app consistently contacts when it is launched. The intuition is that apps are likely to contact the same set of servers at launch time, for example to fetch featured content. Moreover, past work [4, 5] has shown that smart TV apps contact a large number of distinct domains, suggesting that domain access patterns may be a useful fingerprint for smart TV apps. We refer to this type of fingerprint as a *domain-based fingerprint* (DBF):

Definition 4.1. The *domain-based fingerprint* (DBF) of app A, $F_D(A)$, is the set S of domains s.t. for every domain d in S, d appears at least once in at least U launch samples of A. The *size* of $F_D(A)$ is the number of domains in S.

Throughout Section 4, we refer to the parameter U (in Definitions 4.1, 4.2, and 4.3) as the *usage threshold*. Let $F(A)$ denote the fingerprint of app A (of any type, i.e., DBF, PBF, or TBF). U controls the trade-off between the size of $F(A)$ and its *reliability*, i.e., the likelihood that $F(A)$ will manifest itself in live traffic of A.

For example, the size of $F_D(A)$ decreases as U increases, as domains that are only contacted occasionally when A is launched (e.g., if the app has cached some of its resources) will then not become part of $F_D(A)$. Intuitively, and as shown in Section 5, the discriminative power of $F_D(A)$ decreases with the size of $F_D(A)$, as a smaller DBF is less likely to be distinct from other DBFs. On the other hand, a large U implies that $F_D(A)$ manifests itself in most live traffic of A, making it more likely that A can be consistently detected.

Throughout the remainder of this paper, and for all three fingerprinting techniques, we choose the most conservative approach: we set $U = L = 10$, where L is the number of launch samples, i.e., we report results for fingerprints that are *always* present.

Domain Extraction. As all launch samples for the same app A are collected back-to-back (see Section 3.2), any on-device DNS caching will significantly reduce the size of S if domains are only extracted from DNS traffic. FingerprinTV therefore constructs S based on domains found in (1) the answers section of DNS responses, (2) the `Host` header field of HTTP requests (if sent as plaintext), and (3) the TLS SNI extension.

4.1.2 Packet-Pair-Based Fingerprints (PBF)

The second fingerprinting technique we consider identifies an app based on packet sizes and packet directions in packet exchanges that consistently occur when the app is launched. The motivation for this technique came from an observation initially made through visual inspection of the packet captures for Roku apps: the Roku communicates with scribe.logs.roku.com over TLS *every time any app is launched*, and the size of one client-to-server packet in this communication appears to be correlated with the number of digits in the launched app's ID (e.g., the packet's size is 881 bytes if the app ID is a two-digit number, 882 bytes if the app ID is a three-digit number etc.). While the immediate implication of this observation is that Roku is likely tracking the user's app usage, this packet exchange also enables an in-network observer to infer the number of digits in the ID of the app that is launched. If an app exhibits other such consistently occurring packet exchanges, it may be possible to identify the app from observing these packet exchanges happening jointly.

A similar observation was made in our prior work [32], where we introduced the concept of *packet-level signatures* (PLS) for smart home devices. A smart home device exhibits a PLS if the invocation of some specific functionality consistently results in packet exchanges between the device and some endpoint(s), where the packets' sizes (with slight variations) and directions stay consistent across all invocations. In [32], we detail a multi-step methodology for extracting a PLS, where the first step separates packets in TCP connections into *packet pairs*, and later steps reassemble adjacent, consistently occurring packet pairs into longer packet sequences and enforce inter-sequence temporal ordering. Informally, a packet pair is two sequential packets that go in opposite directions, or a single packet

paired with a nil value, if the subsequent packet goes in the same direction; see [32] for the formal definition.

In the technique considered here, we essentially terminate the PLS methodology early, namely when it has identified the consistently occurring packet pairs. This set of packet pairs then constitutes the fingerprint, referred to as a *packet-pair-based fingerprint* (PBF):

Definition 4.2. The *packet-pair-based fingerprint* (PBF) of app A, $F_P(A)$, is the set S of packet pairs such that for every packet pair p in S, p appears U times across L launch samples of A. The *size* of $F_P(A)$ is the number of packet pairs in S.

We use the PingPong tool [32] to extract the consistently occurring packet pairs from our dataset; see Figure 1. We treat the L launch samples of each smart TV app A as corresponding to the smart home events that are triggered L times in [32]. To convert our dataset to the format expected by PingPong, we concatenate the L launch samples of A into one. Then, given that trace as input, PingPong produces clusters of packet pairs of similar sizes and matching directions.

We use the default parameters given in [32], with two modifications. First, we only consider packet pairs with identical packet sizes as candidates for inclusion in $F_P(A)$ (i.e., clusters without any variability in the packet pair sizes), while PingPong allows for small variations in packet sizes in the same cluster. We make this conservative choice to align design choices for PBFs with DBFs: domains used in DBFs have no variation. However, less conservative choices can also be accommodated by our methodology, as discussed in Appendix A. Second, we do not attempt to temporally order packet pairs to create longer packet sequences, as the traffic profiles of smart TV apps are more complex than those of the simpler smart home devices studied in [32]. In particular, there is often a causal explanation for the temporal order of packet sequences in PLS: the device first receives a control command, e.g., "turn off", in one packet sequence, and then updates the cloud with its new state in another packet sequence [32]. On the other hand, a smart TV app may parallelize resource downloads, which makes the temporal order of packet pairs on different connections less predictable.

4.1.3 TLS-Based Fingerprints (TBF)

The third fingerprinting technique we consider attempts to identify an app based on the set of TLS fingerprints the app consistently exhibits when it is launched. We refer to this type of fingerprint as a *TLS-based fingerprint* (TBF). A TBF is conceptually identical to a DBF (see Definition 4.1), but where individual TLS fingerprints assume the role of domains. A TLS fingerprint, originally due to Ristić [45, 46], is the concatenation of a subset of the information that is contained in the TLS Client Hello message that the client sends to the server in order to initiate a TLS session. Since Ristić's work, various implementations have surfaced [80–82]. These mainly differ in terms of what components of the Client Hello they include in the TLS fingerprint. We opt for Mercury [82] because it considers the most comprehensive set of Client Hello components (see [25] for the formal definition), which, presumably, increases the TLS fingerprints' discriminative power. For consistency, we formalize TBFs in Definition 4.3.

Definition 4.3. The *TLS-based fingerprint* (TBF) of app A, $F_T(A)$, is the maximal set S of TLS fingerprints such that for every TLS fingerprint s in S, s appears at least once in at least U launch samples of A. The *size* of $F_T(A)$ is the number of TLS fingerprints in S.

4.2 Fingerprint Performance Assessment

In this section, we define a methodology that forms the basis for how FingerprinTV assesses the performance of a fingerprinting technique F (see the Post Processing box in Figure 1). For F to enable reliable identification of app A, the fingerprint $F(A)$ that F extracts for A must be unique among all other apps' fingerprints. Now, recall from Definitions 4.1, 4.2, and 4.3 in Section 4.1 that at their core, the three types of fingerprints we consider in this paper are essentially just sets with different types of members, namely domains, packet pairs, and TLS fingerprints. Thus, performance assessment of F is fundamentally a set difference problem.

We tackle this problem using agglomerative clustering [83], as it enables us to compute the (dis)similarity of individual apps' fingerprints (i.e., fingerprint member sets) in a structured, yet extensible, way. More precisely, when performed as described below, agglomerative clustering enables us to (1) identify apps that have *distinct* fingerprints, i.e., when the set of fingerprint members that make up $F(A)$ is different from *all* the sets of fingerprint members of all other apps; and (2) identify apps that share the same fingerprint, i.e., when the set of fingerprint members that make up $F(A)$ is identical to the

set of fingerprint members that make up the fingerprint, $F(B)$, of some other app B.

Clustering Procedure. The agglomerative clustering is performed as follows. We first form a fingerprint-member-by-app matrix M such that $M[m_i, A_j]$ holds the number of launch samples of app A_j that fingerprint member m_i was observed in. Next, M is pruned by dropping all rows (fingerprint members) that are not present in at least U (the usage threshold, see Definitions 4.1, 4.2, and 4.3) launch samples for at least one app, i.e., row i is removed *iff* $M[m_i, A_j] < U$ for every j. M is then converted to a binary matrix by setting $M[m_i, A_j] = 0$ if $M[m_i, A_j] < U$ and $M[m_i, A_j] = 1$ if $M[m_i, A_j] \geq U$, for all combinations of i and j. In Sections 5 and 6, we use $U = L = 10$ to conservatively report results for fingerprints that are *always* present, i.e., we set U to be equal to the number of launch samples L we perform (see Section 3.3).

Figure 2 shows an example of the matrix M for the DBFs of 10 popular Fire TV apps. A blue cell indicates that the cell's value is 1, which means that the domain appeared in all $U = 10$ launch samples of the respective app and is therefore part of that app's DBF. A white cell indicates that the cell's value is 0, i.e., the domain appeared in less than 10 launches of the respective app and is therefore *not* part of that app's DBF. The DBF of an app is the binary vector of the corresponding column. For example, the "Facebook" app contacts three domains and we say it has a DBF of size 3; see Definition 4.1. The example also illustrates that DBFs can vary significantly in size: the DBF of "ES File Explorer File Manager" contains a single domain, while the DBF of "NBC" contains 24 domains.

We then compute the agglomerative clustering of the columns (i.e., apps) in M using SciPy [84]. We use cosine distance, defined as $1 - \frac{a \cdot b}{\|a\|_2 \|b\|_2}$, where $a \cdot b$ is the dot product of a and b, the fingerprint member vectors for apps A and B respectively, and $\|x\|_2$ is the 2-norm of x, to compute the distance between two apps [85]. For example, we can see in Figure 2 that the "Lifetime", "A&E", and "HISTORY" apps exhibit the same DBF and, thus, the distance between these apps is 0, and they will be in the same cluster. Since the values in the fingerprint member vectors for apps A and B are binary, a fingerprint member m_i decreases the cosine distance *iff* m_i is part of *both* $F(A)$ *and* $F(B)$. Notice that this implies that the cosine distance will be > 0 when one fingerprint is a subset of another. We consider such fingerprints distinct from one another, since fingerprint subsumption can be accounted for by enforcing timing constraints when examining live traffic.

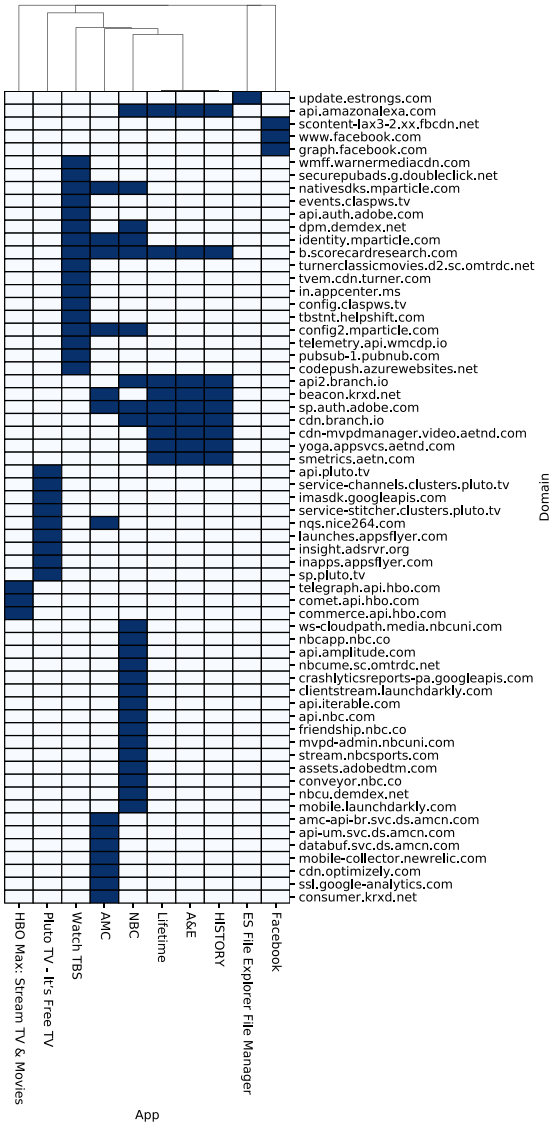

Fig. 2. Example: DBFs of 10 popular Fire TV apps. Rows correspond to domains, columns correspond to apps, and the dendrogram on top corresponds to the clustering of apps based on the similarity of their DBF. A blue cell indicates that the domain is contacted $U = 10$ times and, thus, part of the respective app's DBF; a white cell indicates otherwise. The DBF of an app is the binary vector of the corresponding column. For example, the "Facebook" app has a DBF of size 3 and it is part of a cluster with size 1. The "HISTORY", "A&E", and "Lifetime" apps contact the same nine domains. This means that they have the exact same DBF of size 9, they have distance 0 from each other, and are together in a cluster of size 3.

We use the Nearest Point Algorithm for computing the inter-cluster distances when merging clusters [84] (but the Farthest Point and UPGMA Algorithms pro-

Platform	DBF		PBF		TBF		DBF or PBF		DBF and PBF	
	Prevalence	Distinct.	Prevalence	Distinct.	Prevalence	Distinct.	Prevalence	Distinct.	Prevalence	Distinct.
Apple TV	96%	59%	68%	77%	95%	3%	96%	78%	68%	89%
Fire TV	88%	63%	95%	88%	86%	7%	99%	89%	85%	95%
Roku	100%	46%	100%	72%	100%	1%	100%	76%	100%	76%

Table 1. Summary of the three fingerprinting techniques' performance on the top-1000 apps of the three smart TV platforms. Prevalence is the percentage of apps among the top-1000 that exhibit a fingerprint. Distinctiveness (Distinct.) is the percentage of apps that exhibit a fingerprint that is distinct from all other apps' fingerprints of the same type, among the total number of apps that exhibit a fingerprint of that type (i.e., each distinctiveness column is computed using the raw numbers behind the prevalence percentage values immediately to its left as the baseline).

duce similar results). When extracting clusters from the agglomerative clustering, we use a distance threshold $t = 0$ [86] such that $F(A)$ and $F(B)$ have to be identical to end up in the same cluster. The choice of $t = 0$ also ensures that if $F(A)$ is distinct among all other fingerprints, it will end up in a singleton cluster, and the number of distinct fingerprints is thus simply the number of singleton clusters formed.

We note that we make conservative choices for the parameters in our methodology (e.g., U and t). Even under these strict choices, fingerprints are highly prevalent and have significant discriminative power. Furthermore, the methodology is flexible enough to accommodate less conservative choices, e.g., one can use $U < L = 10$ when extracting DBFs to make a trade-off between its size and reliability, as described in Section 4.1.1.

5 Fingerprinting Results

In this section, we use FingerprinTV to assess the performance of the three fingerprinting techniques introduced in Section 4.1, when applied to the top-1000 Apple TV, Fire TV, and Roku apps (see Section 3.3). Section 5.1 reports the prevalence, distinctiveness, and sizes of the extracted fingerprints. Section 5.2 examines why some apps have identical fingerprints.

5.1 Prevalence, Distinctiveness, and Sizes

This section reports the *prevalence*, *distinctiveness*, and *sizes* of the fingerprints extracted from the dataset described in Section 3.3 using the fingerprinting techniques defined in Section 4.1. We first introduce the three terms and how FingerprinTV computes them, and then proceed to report the numbers for each smart TV platform. The results are summarized in the left part of Table 1.

Prevalence. The *prevalence* of a fingerprint type is the percentage of smart TV apps (from a single platform) that exhibit that type of fingerprint. Recall from Section 4.2 that apps with distinct fingerprints end up in singleton clusters, that apps that have identical fingerprints end up in the same cluster, and that apps that do not exhibit a fingerprint are discarded during clustering. The number N of apps that exhibit a fingerprint is thus the sum of the number of members (i.e., apps) of all clusters in the clustering. The prevalence P is then $P = \frac{N}{1000} \times 100\%$ (as there are 1000 apps per platform).

Distinctiveness. The percentage of apps with distinct fingerprints is arguably the most important metric for assessing how well a fingerprinting technique works. We use the term *distinctiveness* to refer to this metric. As explained in Section 4.2, if the fingerprint, $F(A)$, of app A is distinct, A will end up in a singleton cluster, and the number M of apps with distinct fingerprints is thus equal to the number of clusters with size $x = 1$. The distinctiveness D is then $D = \frac{M}{N} \times 100\%$, i.e., the percentage of apps that exhibit a distinct fingerprint of a given type, taken among all apps that exhibit a fingerprint of that type. A large D thus means that the fingerprinting technique produces fingerprints that are generally able to uniquely identify apps without ambiguity, but is only meaningful if the prevalence is also high.

Sizes. Recall from Definitions 4.1, 4.2, and 4.3 that the size of a fingerprint is the number of fingerprint members it contains, e.g., the number of domains in a DBF. We report the fingerprint sizes to give insights as to how many fingerprint members the general fingerprint contains, and to test the intuition that a larger fingerprint is more likely to be distinct (see Section 4.1.1).

5.1.1 Domain-Based Fingerprints (DBF)

Prevalence. Figures 3a, 3b, and 3c show the number of clusters, grouped by cluster size, for DBFs for the

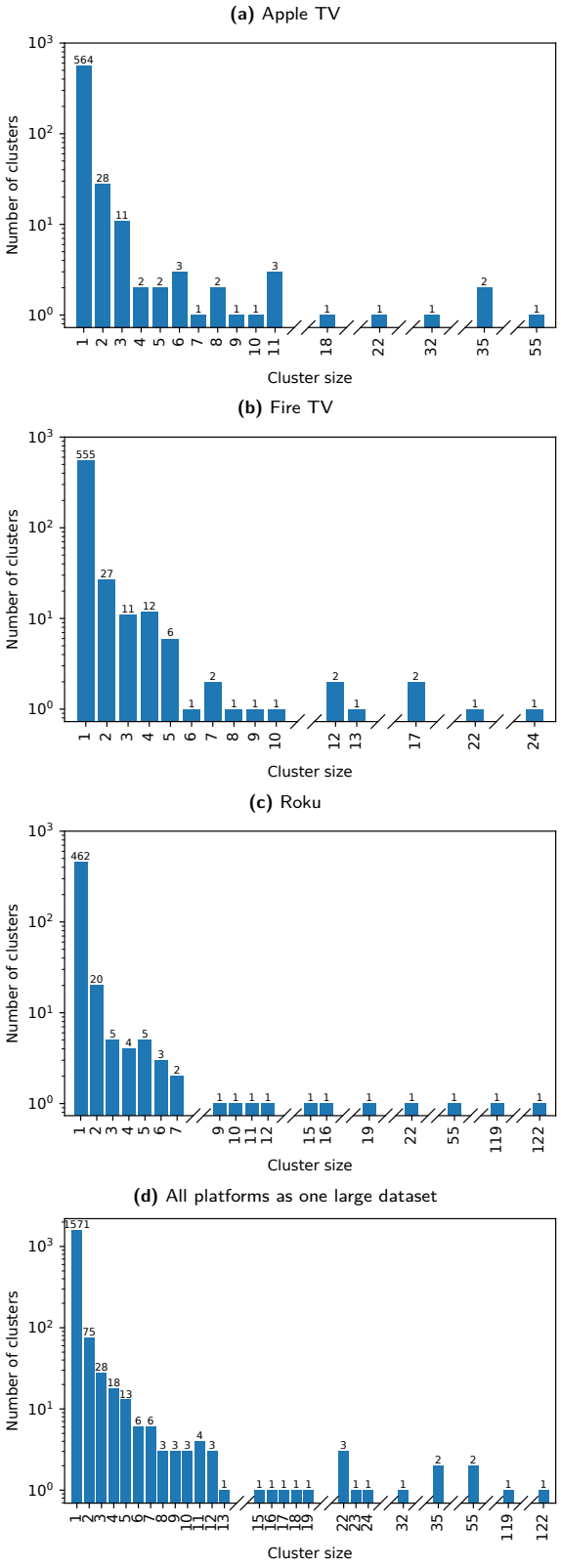

Fig. 3. Distribution of clusters by cluster size for DBFs. The cluster size is the number of apps in a cluster (i.e., apps with the exact same DBF; see Section 4.2). For instance, the bar at $x = 2$ in Figure 3a indicates that there are 28 clusters that each contains 2 apps, for a total of 56 apps.

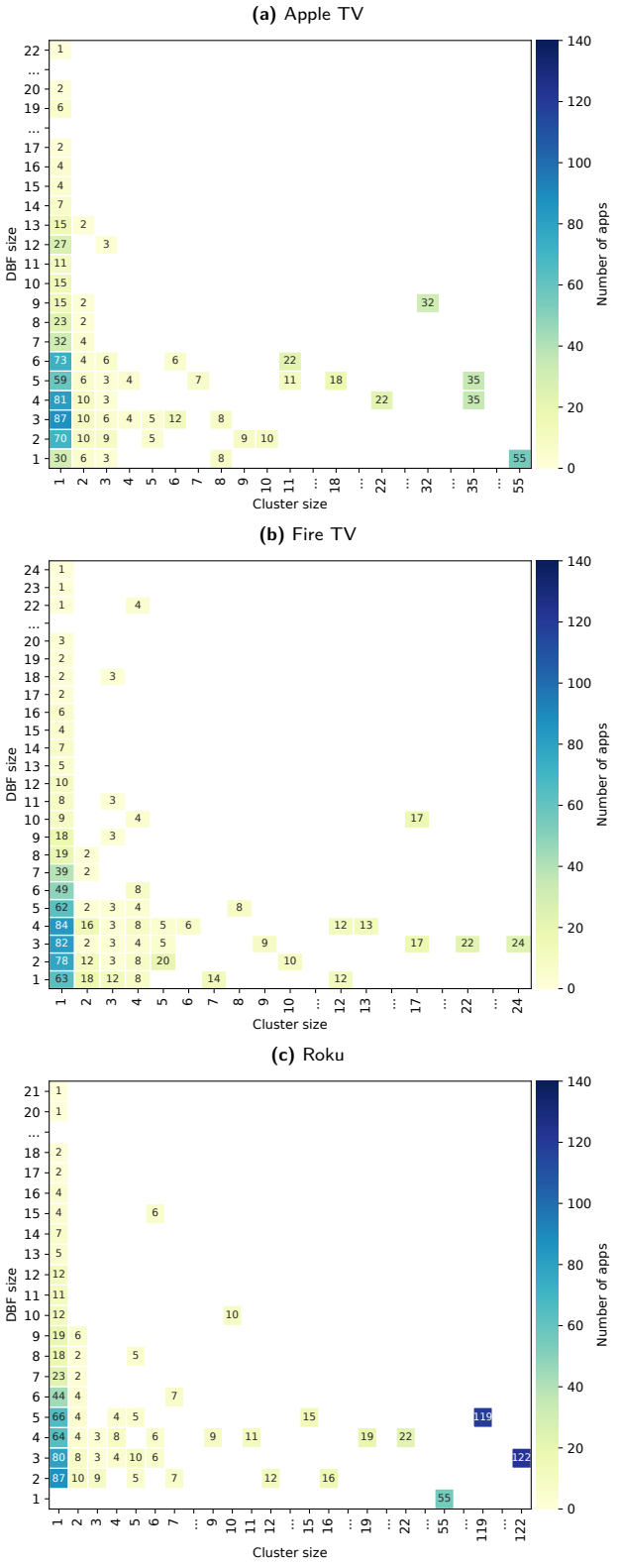

Fig. 4. Distribution of DBF sizes per cluster size. The DBF size is the number of domains in a DBF. The cluster size is the number of apps in a cluster. App counts are shown for each point. For instance, the point at (2, 2) in Figure 4a indicates that there are 10 apps that each has a DBF that contains 2 domains, and these 10 apps reside in clusters that each contains 2 apps.

three smart TV platforms (Figure 3d is discussed in Section 6.1.2). We find that 96% ($N = 961$) of the top-1000 Apple TV apps exhibit a DBF; 88% ($N = 884$) of the top-1000 Fire TV apps exhibit a DBF; and 100% ($N = 1000$) of the top-1000 Roku apps exhibit a DBF.

Distinctiveness. The number of apps with distinct DBFs per platform is the number of clusters with size $x = 1$ in Figures 3a, 3b, and 3c. On all three platforms, the DBF is distinct for about half of the apps that exhibit a DBF: the DBF is distinct for 564 (59%) of the 961 Apple TV apps that exhibit a DBF, 555 (63%) of the 884 Fire TV apps that exhibit a DBF, and 462 (46%) of the 1000 Roku apps that exhibit a DBF.

Sizes. Figure 4 shows the distribution of DBF sizes per cluster size (the label on top of each point is the app count) for the three smart TV platforms. In summary, we find that the median DBF size is four on all three platforms. We also observe that DBF sizes are generally larger for clusters with fewer members, thus the intuition that larger fingerprints are generally more distinct seems to hold true for DBFs.

The three most common DBF sizes are are 4, 5, and 3 for Apple TV; 3, 4, and 2 for Fire TV; and 3, 5, and 4 (tied with 2) for Roku. The median DBF size is 4 across the board. For all three platforms, there appears to be some correlation between a DBF's size and its distinctiveness as the majority of DBFs that are larger than the median DBF are distinct DBFs: 296 of the 463 (64%) Apple TV DBFs, 248 of the 311 (80%) Fire TV DBFs, and 231 of the 420 (55%) Roku DBFs that are larger than the median DBF are distinct.

5.1.2 Packet-Pair-Based Fingerprints (PBF)

Prevalence. Figures 9a, 9b, and 9c (deferred to Appendix B) show the number of clusters, grouped by cluster size, for PBFs for the three smart TV platforms (Figure 9d is discussed in Section 6.1.2). We find that 68% ($N = 678$) of the top-1000 Apple TV apps exhibit a PBF; 95% ($N = 952$) of the top-1000 Fire TV apps exhibit a PBF; and 100% ($N = 1000$) of the top-1000 Roku apps exhibit a PBF.

Distinctiveness. The number of apps with distinct PBFs per platform is the number of clusters with size $x = 1$ in Figures 9a, 9b, and 9c. PBFs have more discriminative power than DBFs: among the apps that exhibit PBFs, 77% of Apple TV apps, 88% of Fire TV apps, and 72% of Roku apps exhibit distinct PBFs.

Sizes. Figure 10 (deferred to Appendix B) shows the distribution of PBF sizes per cluster size (the label on top of each point is the app count) for the three smart TV platforms. The median PBF sizes are 2, 4, and 5 for Apple TV, Fire TV, and Roku, respectively. Like for DBFs, we also observe a strong correlation between a PBF's size and its distinctiveness: 270 of the 272 (99%) Apple TV PBFs, all of the 403 (100%) Fire TV PBFs, and 444 of the 472 (94%) Roku PBFs that are larger than the median PBF are distinct.

5.1.3 TLS-Based Fingerprints (TBF)

We find that TBFs have very little discriminative power and therefore only briefly summarize the results, omitting the diagrams to conserve space. In total, our findings for smart TVs align with those of other work on TLS fingerprinting [25]: (sets of) TLS fingerprints are not sufficiently unique on their own to fingerprint apps.

Prevalence and Sizes. TBFs are about as widespread as DBFs: 95%, 86%, and 100% of the top-1000 Apple TV, Fire TV, and Roku apps, respectively, exhibit a PBF. The median TBF size is 2 across all platforms. While prevalence is large for TBFs, this also bears the positive message that most smart TV apps encrypt (part of) their communication. The TBF prevalence observed for Roku is in line with what is reported in the appendix of [4], but we observe slightly fewer TBFs for Fire TV, possibly because we only consider "on-launch" traffic, whereas [4] also inject user actions post launch.

Distinctiveness. While TBFs are prevelant on all three platforms, they have little discriminative power: only 3%, 7%, and 1% of the Apple TV, Fire TV, and Roku apps that exhibit TBFs, exhibit distinct TBFs. Furthermore, a few TBFs are shared by a large number of apps; e.g., for both Apple TV and Roku, the clustering outputs two clusters with over 300 apps.

5.1.4 Takeaways

DBFs, PBFs, and TBFs are all prevalent among Apple TV, Fire TV, and Roku apps. However, only DBFs and PBFs have enough discriminative power to reliable identify apps, and we therefore omit TBFs from the discussion going forward. We also observe a correlation between a fingerprint's size and its discriminative power. Overall, PBFs seem to have more discriminative power than DBFs, but they are also more likely to change

over time: even small changes to an app's communication protocol(s) directly affect packet sizes [32]. In Appendix B.1, we provide additional details on DBFs' and PBFs' distinctiveness by considering increasingly larger portions of each platform's dataset.

5.2 Identical Fingerprints

This section examines to what extent apps with identical fingerprints stem from the same developer. If a developer releases multiple apps for the same platform, they may opt to use some of the same backend servers to deliver content, which could make these apps exhibit identical fingerprints. We investigate this by examining the number of distinct developers present in clusters with size $x > 1$, i.e., clusters of apps that share the same fingerprint. We consider developers to be identical if they are part of the same parent organization. For example, the Fire TV app developers "Scripps Networks, LLC", "Discovery Communications" and "OWN, LLC" are identical, since Discovery, Inc. owns a majority stake in these companies.

5.2.1 Domain-Based Fingerprints

Summary. Figure 5 shows the number of clusters of size x (for DBFs) that contain apps from Q distinct developers for the three smart TV platforms. In summary, we find that a sizeable fraction of identical DBFs are indeed attributable to apps from the same developer. We also find many examples of apps that appear to be generated using "no code" toolkits provided by consulting firms, and apps that are generated using the same toolkit tend to have identical DBFs.

Apple TV Details. For Apple TV, 142 of the 397 apps (37%) that share their DBF with other app(s) only share it with other apps from the same developer. We note that while the apps in the two clusters of size $x = 35$ in Figure 5a are officially published by completely different developers ($Q = 35$ for these clusters), they all appear to be due to the same consulting firm, Subsplash, Inc., (who offers a "no code" toolkit for generating and publishing apps) since their DBFs contain subdomains of subsplash.com and (most of) their bundle IDs begin with "com.subsplash". Further, they are all religious apps, which is Subsplash's specialty. Similarly, all apps in one of the three clusters of size $x = 11$, in one of the two clusters of size $x = 4$, and in one of the 28 clusters of size $x = 2$ appear to be due to MAZ Systems

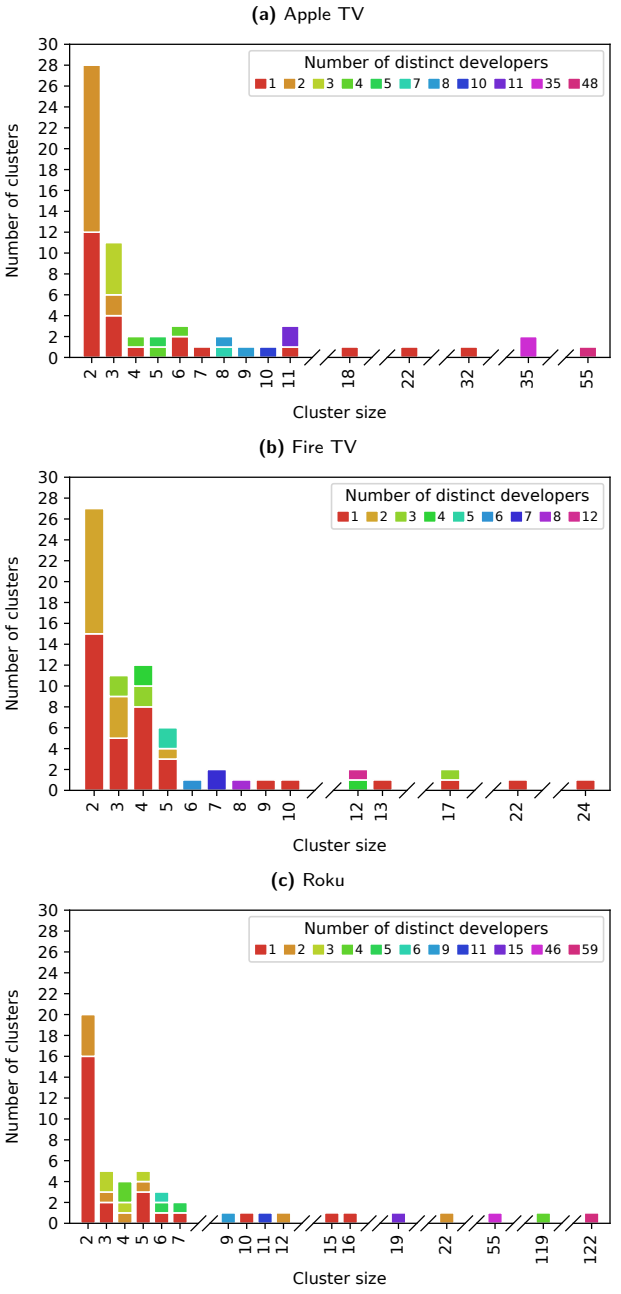

Fig. 5. Distribution of the number of developers responsible for apps in clusters of size $x > 1$ for DBFs. For instance, the bar at $x = 2$ in Figure 5a indicates that 12 of the 28 clusters of size $x = 2$ (see Figure 3a) only contain apps from the same developer, while the remaining 16 contain apps from 2 developers.

Inc.; all apps in one of the two clusters of size $x = 8$ and in one of the 28 clusters of size $x = 2$ appear to be due to UscreenTV, LLC; and RadioKing and Streann Media Inc each appear to be behind all apps in one of the 28 clusters of size $x = 2$. If we treat these cases as the same developer, we instead get that 243 of the 397

apps (61%) that share their DBF with other app(s) only share it with other apps from the same developer.

Fire TV Details. For Fire TV, 187 of the 329 apps (57%) that share their DBF with other app(s) only share it with other apps from the same developer. We observe that when a DBF is shared among many apps, those apps generally stem from the same developer: notice that $Q = 1$ for many of the clusters of size $x > 8$ in Figure 5b. We note that we suspect that the $Q = 3$ distinct developers responsible for the apps in one of the two clusters of size $x = 17$ are the same (or closely related) entities as these apps (1) have custom made, yet similar, privacy policies; (2) use the same pattern for their contact emails; and (3) are all ambience apps that use the same app naming scheme.

As for Apple TV, we again find many clusters of apps that are officially published by multiple developers (and shown as such in Figure 5b), but where the domains in their DBFs and/or their package names suggest that they are from the same consulting firms. If we treat these cases as the same developer, 227 of the 329 apps (69%) that share their DBF with other app(s) only share it with other apps from the same developer.

Roku Details. For Roku, 107 of the 538 apps (20%) that share their DBF with other app(s) only share it with other apps from the same developer. We again find many clusters of apps that are officially published by multiple developers (and shown as such in Figure 5c), but where the domains in the DBFs suggest that they are due to the same consulting firms. If we treat these cases as the same developer, 143 of the 538 apps (27%) that share their DBF with other app(s) only share it with other apps from the same developer.

5.2.2 Packet-Pair-Based Fingerprints

We do not find evidence that identical PBFs primarily stem from apps from the same developer. For Apple TV, 10 of the 153 apps (16%) that share their PBF with other app(s) only share it with other apps from the same developer. This also only applies to 7 of 119 (6%) Fire TV apps, and 42 of 283 (15%) Roku apps.

5.2.3 Takeaways

When multiple apps exhibit identical fingerprints, an in-network observer can only make ambiguous inferences about app usage. However, apps with identical DBFs primarily stem from the same developer. As some developers focus their efforts on building apps with a certain theme (e.g., religious apps), it may thus still be possible to infer what *type of content* the user is consuming. The same does not hold true for PBFs, but identical PBFs are less widespread (see Section 5.1.2).

6 Mix & Match Fingerprints

This section provides a cross-platform analysis of fingerprints and examines combined use of fingerprints. Section 6.1 compares DBFs and PBFs across the three smart TV platforms. Section 6.2 examines to what extent combining DBFs and PBFs improves discriminative power. We omit TBFs from the discussion because of their poor performance (see Section 5.1.3).

6.1 Fingerprints Across Platforms

This section compares how fingerprints of apps that are present on all three platforms differ across platforms and extends the evaluation in Section 5.1 of the extracted fingerprints' distinctiveness by considering the datasets of the three platforms as a single, large dataset.

6.1.1 Multi-Platform Apps

We first compare the DBFs and PBFs of apps in our dataset that are present on all three platforms, referred to as *multi-platform apps*. Drawing inspiration from [5], we identify 80 multi-platform apps through fuzzy matching on the app and developer names, as the same app can have slightly different names on each platform, e.g., "YouTube" on Fire TV and Roku, but "YouTube: Watch, Listen, Stream" on Apple TV.

DBFs. Figure 6 shows, using one color-coded bar per platform per app, the sizes of the DBFs of 60 of the 80 multi-platform apps; due to space constraints, we only show the 60 multi-platform apps with the largest DBFs in descending order. The textured part of each bar indicates domains in the DBF that are unique to the corresponding platform, i.e., domains that are not present in the DBFs of the same app on the other two platforms. Most multi-platform apps have at least one platform-specific domain in their DBFs for all three versions of the app (Apple TV, Fire TV, and Roku). In fact, only 19 of the 80 multi-platform apps (24%) have version(s)

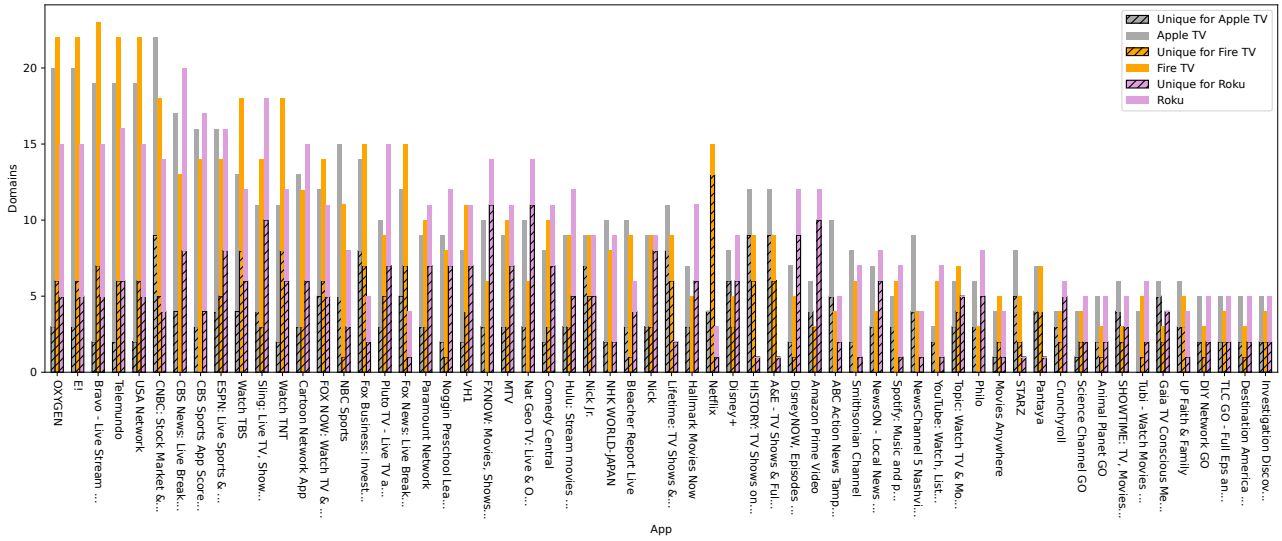

Fig. 6. DBF sizes for the 60 multi-platform apps that exhibit the largest DBFs in descending order. The size of the DBF for each version (Apple TV, Fire TV, and Roku) of an app is indicated using color-coded bars. The textured part of each bar indicates domains in the DBF that are unique to the corresponding platform.

with no unique domains in their DBFs, and in 18 cases this only applies to a single version of the app. Interestingly, it is primarily the Fire TV version that lacks platform-specific domains in its DBF. The Roku version *always* has at least one unique domain in its DBF, as *all* Roku apps communicate with scribe.logs.roku.com when they are launched (see Section 4.1.2).

PBFs. All apps, except the Apple TV version of two apps, have platform-specific packet pairs in their PBFs for all versions of the app (we defer the PBF variant of Figure 6 to Figure 11 in Appendix C). In fact, the vast majority—and in many cases *all*—packet pairs in the PBFs are platform-specific.

6.1.2 Distinctiveness of Fingerprints Across Platforms

To further assess the distinctiveness of fingerprints across platforms, we perform a similar analysis as in Section 5.1, but where we consider fingerprints of apps on all three platforms as a single, large dataset. We refer to apps with the same fingerprint as having a "collision".

DBFs. Figure 3d shows the number of clusters, grouped by cluster size, when the DBFs of apps of all three platforms are considered as a single, large dataset. The impact this merging of datasets has on the DBFs' distinctiveness is understood by comparing Figure 3d and the sum of Figures 3a, 3b, and 3c. If the DBFs are less distinct in the merged dataset, the bar corresponding to the number of clusters with size $x = 1$ in Figure 3d will be smaller than the sum of the corresponding bars at $x = 1$ in Figures 3a, 3b, 3c. Similarly, increases in DBF collisions would be reflected as larger values in Figure 3d compared to the sum of the values at the respective x in Figures 3a, 3b, and 3c, for $x > 1$.

We observe a very slight decrease in the number of distinct DBFs: there are $564 + 555 + 462 = 1581$ distinct DBFs when the platforms are considered individually (Figures 3a, 3b, and 3c), and 1571 distinct DBFs when the three platforms are considered together (Figure 3d). DBF collisions also only change slightly: the number of clusters with size $x > 1$ are mostly the same across Figure 3d and the sum of Figures 3a, 3b, and 3c. We note that *all* additional DBF collisions that arise in the merged dataset are among Apple TV and Fire TV apps. This is because *all* Roku apps' DBFs include one Roku domain (see Section 4.1.2). Furthermore, the collisions are attributable to DBFs that are smaller than or equal to the median DBF size, which further confirms the intuition that larger DBFs have more discriminative power (see Sections 4.1.1 and 5.1.1).

PBFs. Due to space constraints, we defer the PBF analog of Figure 3d to Figure 9d in Appendix C. We observe *no change* in the number of distinct PBFs when the three individual datasets are considered as one: the sum of distinct fingerprints across Figures 9a, 9b, and 9c is 2075, which is also the number of distinct fingerprints in Figure 9d. There is also hardly any change to PBF collisions as the number of clusters with size $x > 1$ are almost identical across Figure 9d and the sum of Fig-

ures 9a, 9b, and 9c. In fact, the only change is that a cluster of two Fire TV apps is merged with a cluster that contains 12 Apple TV apps. The PBFs of these apps are comprised of a single MTU-sized client-to-server packet, likely because the client is sending data in bulk. This results in single-packet pairs (see Section 4.1.2 and [32]), which have little discriminative power.

6.1.3 Takeaways

The DBFs and PBFs of apps that are made available on all three platforms are often platform-specific. Thus, it is generally possible to not only fingerprint smart TV apps themselves, but also (as a side-effect) to identify which smart TV platform they are being used on. Additionally, the distinctiveness of DBFs and PBFs appears to hold steady when the three datasets are considered as one large dataset. This provides some indication that the fingerprints are likely to retain their discriminative power in an open world setting.

6.2 Combining Fingerprints

This section considers the benefits of combining DBFs and PBFs. To establish to what extent smart TV apps can be fingerprinted in general, using any technique, we first examine how many apps exhibit a DBF or a PBF (or both). Since larger fingerprints have more discriminative power (as confirmed in Section 5.1.4), we examine how many apps exhibit *both* a DBF *and* a PBF.

DBF or PBF. The "DBF or PBF" column of Table 1 lists the prevalence and distinctiveness of fingerprints that are comprised of a DBF or a PBF (at least one, or both). The reported distinctiveness is based on the distinctiveness of the individual components considered separately. That is, a fingerprint is *not* considered distinct if neither the DBF nor the PBF are distinct on their own, even if the combination of them is exclusive to one app. This is to remain conservative in our reporting, as confusion could arise if another app that exhibits the same DBF is launched at the same time as a third app that exhibits the same PBF, e.g., on different smart TVs in the same household.

The results show that nearly all apps on all three platforms exhibit either a DBF or a PBF (or both): the prevalence is $\geq 96\%$ across the board. More importantly, most of these fingerprints are distinct: the fingerprint is distinct for 78% (750) of the Apple TV apps, 89% (884) of the Fire TV apps, and 76% (760) of the Roku apps that exhibit a DBF or a PBF (or both). It is worth noting that even for Apple TV, where DBFs contribute significantly to the prevalence (PBFs on their own only achieve 68% prevalence, whereas this joint fingerprint achieves 96%), the distinctiveness is in line with that of PBFs on their own (recall from Section 5.1.4 that PBFs have more discriminative power than DBFs). Thus, the extra coverage that can be achieved by also considering DBFs appear to primarily stem from distinct DBFs.

DBF and PBF. The "DBF and PBF" column of Table 1 lists the prevalence and distinctiveness of fingerprints that are comprised of *both* a DBF *and* a PBF. The reported distinctiveness is computed in the same way as for "DBF or PBF", described above.

The results show that DBFs and PBFs generally co-occur: notice that the prevalence for this joint fingerprint almost equals its upper bound, i.e., the minimum prevalence of DBFs and PBFs individually. This implies that almost all Apple TV apps that exhibit a PBF must also exhibit a DBF, and vice versa for Fire TV (*all* Roku apps exhibit both DBFs and PBFs). The fingerprints also have high discriminative power: the fingerprint is distinct for 89% (599) of the Apple TV apps, 95% (802) of the Fire TV apps, and 76% (760) of the Roku apps that exhibit *both* a DBF *and* a PBF.

6.2.1 Takeaways

Nearly *all* apps across all three smart TV platforms can be fingerprinted if one uses either a DBF or a PBF (or both) on a per-app basis. Moreover, these joint fingerprints have high discriminative power, but fingerprints for Roku apps fall slightly short of those of Apple TV apps and Fire TV apps in this respect.

7 Discussion

This section briefly discusses how to mitigate the inferences that can be made using DBFs and PBFs, as well as the limitations of our work, and future directions.

Possible Defenses. As DBFs rely on access to domain names in cleartext, DNS-over-HTTPS (DoH) [87] and DNS-over-TLS (DoT) [88] are perhaps the most obvious potential defenses. However, as evident from the results reported in Section 5.1.3, most smart TV apps consistently communicate over TLS at every launch. If the smart TV app uses the TLS Server Name Indication (SNI) [89] extension for these sessions, the domain can

be recovered from the TLS session alone, which negates the defense provided by DoH/DoT. To test this, we ran a modified version of FingerprinTV that would disregard all DNS data, and confirmed that the results were almost identical to those reported in Section 5.1.1. In essence, to be effective against DBFs, DoH/DoT must be paired with Encrypted SNI (ESNI), but this requires that the apps' backend servers add support for ESNI, which likely lies years ahead.

Network-level blockers, such as Pi-hole, that block traffic to select domains can prevent identification of an app via its DBF, if the DBF includes one or more domains from the blocklist, as the adversary will only observe a partial DBF match. The adversary can easily counter such protection by removing domains that are present in popular blocklists from the DBF they extract for the app during training, and then (also) examine live traffic for the manifestation of this reduced DBF. Since smaller DBFs have less discriminative power (see Sections 4.1.1 and 5.1.4), network-level blockers may add uncertainty to the adversary's inferences, but may also introduce app breakage.

For complete protection against DBFs, the smart TV's traffic can be tunneled through a VPN. Assuming the adversary is somewhere on the path from the smart TV to the VPN server, this defense nullifies the effectiveness of DBFs entirely, as the tunnel encrypts the three protocol fields that domains can be extracted from (see Section 4.1.1). The downsides are that tunneling adds additional network overhead, and that the VPN server may become a bottleneck (e.g., if it enforces per-user rate limits that the smart TV can saturate).

For defenses against PBFs, we refer to our prior work [32]. There, we argue that packet padding is effective against fingerprints that rely on packet sizes, at the cost of some overhead, which also applies to the special case of PBFs considered in this paper. In summary, a VPN that pads packets provides an effective defense against inference attacks based on DBFs (by hiding the destination) and/or PBFs (by obfuscating packet sizes).

Limitations. While our assessment of the feasibility of fingerprinting smart TV apps is made against the largest smart TV dataset to date, it is not without limitations. In particular, some smart TV apps require that the user logs in, e.g., subscription-based apps such as Netflix. As FingerprinTV does not attempt to create accounts and log in, the fingerprints it extracts for these apps may be different from what can be observed in the wild as the on-launch traffic may differ depending on the login state. This is a common limitation of automated measurement studies, e.g., [4, 5, 90, 91], as automating account creation and login is a difficult problem. Some studies approach this limitation by manually logging in to a subset of the tested apps and/or by relying on third-party authentication such as Google [90, 92, 93], while others record the key presses involved in the login procedure s.t. future versions of the same app can be tested automatically [94]. An adversary intent on extracting (more) precise fingerprints for select smart TV apps can adopt a similar strategy using FingerprinTV.

Future Directions. In this paper, we purposely opted for the most conservative design choices to extract reliable fingerprints. For example, we require that a domain must be present at least once in *all* launch samples of some app A for it to be included in A's DBF, and, unlike in [32], we do *not* allow for small variations in packet sizes when considering packet pairs for inclusion in a PBF. It may be possible to increase fingerprint prevalence and sizes by relaxing these strict requirements, possibly at the cost of fingerprint reliability and/or distinctiveness. Future work can study these trade-offs by adjusting existing parameters of our FingerprinTV framework (both in pre-processing and in the clustering algorithm).

Since network traffic fingerprints may change over time, it would be interesting to perform a longitudinal analysis to see how the extracted fingerprints evolve, and how often fingerprints need to be extracted. FingerprinTV facilitates such studies through its automated data collection, fingerprint extraction, and fingerprint assessment features: the clustering that is inherently unsupervised in our methodology will identify and extract any updated fingerprints.

8 Conclusion

We presented FingerprinTV, a methodology and implementation we devised for automatically extracting and evaluating network fingerprints of smart TV apps. By deploying FingerprinTV to the top-1000 Apple TV, Fire TV, and Roku apps, we showed that smart TV app fingerprinting is highly feasible and effective on all three platforms, as most apps exhibit a fingerprint, and most fingerprints are distinct. We plan to make the FingerprinTV code and dataset publicly available [47].

Acknowledgments

This work is supported by NSF Awards 1815666 and 1956393. The authors would like to thank our shepherd, Liang Wang, and the anonymous PETS reviewers for their insightful feedback that helped improve the paper.

References

[1] Leichtman Research Group, Inc. 39% of Adults Watch Video via a Connected TV Device Daily. https://www.leichtmanresearch.com/39-of-adults-watch-video-via-a-connected-tv-device-daily/. [Online; accessed 2021-10-27].

[2] Hub Research LLC. 2021 Connected Home. https://hubresearchllc.com/reports/?category=2021&title=2021-connected-home, 2021. [Online; accessed 2021-10-27].

[3] Hub Research LLC. 2021 Evolution of the TV Set. https://hubresearchllc.com/reports/?category=2021&title=2021-evolution-of-the-tv-set, 2021. [Online; accessed 2021-10-27].

[4] Hooman Mohajeri Moghaddam, Gunes Acar, Ben Burgess, Arunesh Mathur, Danny Yuxing Huang, Nick Feamster, Edward W. Felten, Prateek Mittal, and Arvind Narayanan. Watching You Watch: The Tracking Ecosystem of Over-the-Top TV Streaming Devices. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 131–147, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Janus Varmarken, Hieu Le, Anastasia Shuba, Athina Markopoulou, and Zubair Shafiq. The TV is Smart and Full of Trackers: Measuring Smart TV Advertising and Tracking. *Proceedings on Privacy Enhancing Technologies*, 2020(2), 2020.

[6] Andrew Hintz. Fingerprinting websites using traffic analysis. In Roger Dingledine and Paul Syverson, editors, *Privacy Enhancing Technologies*, pages 171–178, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[7] Qixiang Sun, D.R. Simon, Yi-Min Wang, W. Russell, V.N. Padmanabhan, and Lili Qiu. Statistical identification of encrypted web browsing traffic. In *Proceedings 2002 IEEE Symposium on Security and Privacy*, pages 19–30, 2002.

[8] George Dean Bissias, Marc Liberatore, David Jensen, and Brian Neil Levine. Privacy vulnerabilities in encrypted http streams. In George Danezis and David Martin, editors, *Privacy Enhancing Technologies*, pages 1–11, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[9] Marc Liberatore and Brian Neil Levine. Inferring the source of encrypted http connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 255–263, New York, NY, USA, 2006. Association for Computing Machinery.

[10] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: Attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, page 31–42, New York, NY, USA, 2009. Association for Computing Machinery.

[11] Liming Lu, Ee-Chien Chang, and Mun Choon Chan. Website fingerprinting and identification using ordered feature sequences. In Dimitris Gritzalis, Bart Preneel, and Marianthi Theoharidou, editors, *Computer Security – ESORICS 2010*, pages 199–214, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[12] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, page 103–114, New York, NY, USA, 2011. Association for Computing Machinery.

[13] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *2012 IEEE Symposium on Security and Privacy*, pages 332–346, 2012.

[14] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 143–157, San Diego, CA, August 2014. USENIX Association.

[15] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, 2016.

[16] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1187–1203, Austin, TX, August 2016. USENIX Association.

[17] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1928–1943, New York, NY, USA, 2018. Association for Computing Machinery.

[18] Se Eun Oh, Saikrishna Sunkam, and Nicholas Hopper. p1-fp: Extraction, classification, and prediction of website fingerprints with deep learning. *Proceedings on Privacy Enhancing Technologies*, 2019(3), 2019.

[19] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-CNN: A Data-Efficient Website Fingerprinting Attack Based on Deep Learning. *Proceedings on Privacy Enhancing Technologies*, 4:292–310, 2019.

[20] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-Shot Learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1131–1148, New York, NY, USA, 2019. Association for Computing Machinery.

[21] Se Eun Oh, Nate Mathews, Mohammad Saidur Rahman, Matthew Wright, and Nicholas Hopper. GANDaLF: GAN for Data-Limited Fingerprinting. *Proceedings on Privacy Enhancing Technologies*, 2021(2):305–322, 2021.

[22] Jean-Pierre Smith, Prateek Mittal, and Adrian Perrig. Website Fingerprinting in the Age of QUIC. *Proceedings on Privacy Enhancing Technologies*, 2021(2):48–69, 2021.

[23] Xiaobo Ma, Mawei Shi, Bingyu An, Jianfeng Li, Daniel Xiapu Luo, Junjie Zhang, and Xiaohong Guan. Context-aware

website fingerprinting over encrypted proxies. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.

[24] Nguyen Phong Hoang, Arian Akhavan Niaki, Phillipa Gill, and Michalis Polychronakis. Domain name encryption is not enough: privacy leakage via IP-based website fingerprinting. *Proceedings on Privacy Enhancing Technologies*, 2021(4):420–440, 2021.

[25] Blake Anderson and David McGrew. Accurate TLS Fingerprinting using Destination Context and Knowledge Bases, 2020.

[26] Vincent F. Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 439–454, 2016.

[27] Vincent F. Taylor, Riccardo Spolaor, Mauro Conti, and Ivan Martinovic. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Transactions on Information Forensics and Security*, 13(1):63–78, 2018.

[28] Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J Dubois, Martina Lindorfer, David Choffnes, Maarten van Steen, and Andreas Peter. Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, 2020.

[29] Bogdan Copos, Karl Levitt, Matt Bishop, and Jeff Rowe. Is anybody home? inferring activity from smart home network traffic. In *2016 IEEE Security and Privacy Workshops (SPW)*, pages 245–251, 2016.

[30] TJ OConnor, Reham Mohamed, Markus Miettinen, William Enck, Bradley Reaves, and Ahmad-Reza 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*, WiSec '19, page 128–138, New York, NY, USA, 2019. Association for Computing Machinery.

[31] Jingjing Ren, Daniel J. Dubois, David Choffnes, Anna Maria Mandalari, Roman Kolcun, and Hamed Haddadi. Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. In *Proceedings of the Internet Measurement Conference*, IMC '19, page 267–279, New York, NY, USA, 2019. Association for Computing Machinery.

[32] Rahmadi Trimananda, Janus Varmarken, Athina Markopoulou, and Brian Demsky. Packet-Level Signatures for Smart Home Devices. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, February 2020.

[33] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk 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*, WiSec '20, page 207–218, New York, NY, USA, 2020. Association for Computing Machinery.

[34] Sha Zhao, Shijian Li, Julian Ramos, Zhiling Luo, Ziwen Jiang, Anind K. Dey, and Gang Pan. User profiling from their use of smartphone applications: A survey. *Pervasive and Mobile Computing*, 59:101052, 2019.

[35] Edward C. Malthouse, Ewa Maslowska, and Judy U. Franks. Understanding programmatic TV advertising. *International Journal of Advertising*, 37(5):769–784, 2018.

[36] FTC Staff. A Look at What ISPs Know About You: Examining the Privacy Practices of Six Major Internet Service Providers, October 2021.

[37] Alex Sherman. How Roku used the Netflix playbook to beat bigger players and rule streaming video. https://www.cnbc.com/2021/06/18/how-roku-dominated-streaming-anthony-woods-new-content-obsession.html, 2021.

[38] Srinivas Krishnan and Fabian Monrose. DNS Prefetching and Its Privacy Implications: When Good Things Go Bad. In *Proceedings of the 3rd USENIX Conference on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, LEET'10, page 10, USA, 2010. USENIX Association.

[39] Noah Apthorpe, Dillon Reisman, and Nick Feamster. A Smart Home is No Castle: Privacy Vulnerabilities of Encrypted IoT Traffic, 2017.

[40] Hang Guo and John Heidemann. IP-Based IoT Device Detection. In *Proceedings of the 2018 Workshop on IoT Security and Privacy*, IoT S&P '18, page 36–42, New York, NY, USA, 2018. Association for Computing Machinery.

[41] Franck Le, Jorge Ortiz, Dinesh Verma, and Dilip Kandlur. *Policy-Based Identification of IoT Devices' Vendor and Type by DNS Traffic Analysis*, pages 180–201. Springer International Publishing, Cham, 2019.

[42] Roberto Perdisci, Thomas Papastergiou, Omar Alrawi, and Manos 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)*, pages 474–489, 2020.

[43] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *Proceedings of the 2006 ACM CoNEXT Conference*, CoNEXT '06, New York, NY, USA, 2006. Association for Computing Machinery.

[44] Laurent Bernaille and Renata Teixeira. Early recognition of encrypted applications. In Steve Uhlig, Konstantina Papagiannaki, and Olivier Bonaventure, editors, *Passive and Active Network Measurement*, pages 165–175, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[45] Ivan Ristić. HTTP client fingerprinting using SSL handshake analysis. https://blog.ivanristic.com/2009/06/http-client-fingerprinting-using-ssl-handshake-analysis.html.

[46] Qualys SSL Labs. HTTP Client Fingerprinting Using SSL Handshake Analysis. https://www.ssllabs.com/projects/client-fingerprinting/.

[47] UCI Networking Group. FingerprinTV. https://github.com/UCI-Networking-Group/fingerprintv.

[48] Andrew W. Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In Constantinos Dovrolis, editor, *Passive and Active Network Measurement*, pages 41–54, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[49] Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey M. Voelker. Unexpected means of protocol inference. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, IMC '06, page 313–326, New York, NY, USA, 2006. Association for Com-

puting Machinery.

[50] Stanislav Miskovic, Gene Moo Lee, Yong Liao, and Mario Baldi. Appprint: Automatic fingerprinting of mobile applications in network traffic. In Jelena Mirkovic and Yong Liu, editors, *Passive and Active Measurement*, pages 57–69, Cham, 2015. Springer International Publishing.

[51] Xuan Feng, Qiang Li, Haining Wang, and Limin Sun. Acquisitional rule-based engine for discovering internet-of-things devices. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 327–341, Baltimore, MD, August 2018. USENIX Association.

[52] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. Studying TLS Usage in Android Apps. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '17, page 350–362, New York, NY, USA, 2017. Association for Computing Machinery.

[53] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. SoK: Security Evaluation of Home-Based IoT Deployments. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1362–1380, 2019.

[54] Danny Yuxing Huang, Noah Apthorpe, Frank Li, Gunes Acar, and Nick Feamster. IoT Inspector: Crowdsourcing Labeled Network Traffic from Smart Home Devices at Scale. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 4(2), June 2020.

[55] Martin Husák, Milan Čermák, Tomáš Jirsík, and Pavel Čeleda. HTTPS Traffic Analysis and Client Identification Using Passive SSL/TLS Fingerprinting. *EURASIP J. Inf. Secur.*, 2016(1), December 2016.

[56] Kailong Wang, Junzhe Zhang, Guangdong Bai, Ryan Ko, and Jin Song Dong. It's Not Just the Site, It's the Contents: Intra-Domain Fingerprinting Social Media Websites Through CDN Bursts. In *Proceedings of the Web Conference 2021*, WWW '21, page 2142–2153, New York, NY, USA, 2021. Association for Computing Machinery.

[57] Mauro Conti, Luigi Vincenzo Mancini, Riccardo Spolaor, and Nino Vincenzo Verde. Analyzing android encrypted network traffic to identify user actions. *IEEE Transactions on Information Forensics and Security*, 11(1):114–125, 2016.

[58] Brendan Saltaformaggio, Hongjun Choi, Kristen Johnson, Yonghwi Kwon, Qi Zhang, Xiangyu Zhang, Dongyan Xu, and John Qian. Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic. In *10th USENIX Workshop on Offensive Technologies (WOOT 16)*, Austin, TX, August 2016. USENIX Association.

[59] Sean Kennedy, Haipeng Li, Chenggang Wang, Hao Liu, Boyang Wang, and Wenhai Sun. I Can Hear Your Alexa: Voice Command Fingerprinting on Smart Home Speakers. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 232–240, 2019.

[60] Chenggang Wang, Sean Kennedy, Haipeng Li, King Hudson, Gowtham Atluri, Xuetao Wei, Wenhai Sun, and Boyang Wang. 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, page 254–265, New York, NY, USA, 2020. Association for Computing Machinery.

[61] Batyr Charyyev and Mehmet Hadi Gunes. Voice command fingerprinting with locality sensitive hashes. In *Proceedings of the 2020 Joint Workshop on CPS&IoT Security and Privacy*, CPSIOTSEC'20, page 87–92, New York, NY, USA, 2020. Association for Computing Machinery.

[62] Jack Hyland, Conrad Schneggenburger, Nick Lim, Jake Ruud, Nate Mathews, and Matthew Wright. What a SHAME: Smart Assistant Voice Command Fingerprinting Utilizing Deep Learning. In *Proceedings of the 20th Workshop on Privacy in the Electronic Society*, WPES '21, page 237–243, New York, NY, USA, 2021. Association for Computing Machinery.

[63] Andrew Reed and Michael Kranch. Identifying HTTPS-Protected Netflix Videos in Real-Time. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, page 361–368, New York, NY, USA, 2017. Association for Computing Machinery.

[64] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the Burst: Remote Identification of Encrypted Video Streams. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1357–1374, Vancouver, BC, August 2017. USENIX Association.

[65] Said Jawad Saidi, Anna Maria Mandalari, Roman Kolcun, Hamed Haddadi, Daniel J. Dubois, David Choffnes, Georgios Smaragdakis, and Anja Feldmann. A Haystack Full of Needles: Scalable Detection of IoT Devices in the Wild. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 87–100, New York, NY, USA, 2020. Association for Computing Machinery.

[66] Janus Varmarken, Hieu Le, Anastasia Shuba, Athina Markopoulou, and Zubair Shafiq. Firetastic. https://github.com/UCI-Networking-Group/firetastic, 2020.

[67] Janus Varmarken, Hieu Le, Anastasia Shuba, Athina Markopoulou, and Zubair Shafiq. Rokustic. https://github.com/UCI-Networking-Group/rokustic, 2020.

[68] Girard Kelly, Jeff Graham, Jill Bronfman, and Steve Garton. Privacy of Streaming Apps and Devices: Watching TV that Watches Us. San Francisco, CA: Common Sense Media, 2021.

[69] Marco Ghiglieri and Erik Tews. A privacy protection system for HbbTV in Smart TVs. In *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*, pages 357–362, 2014.

[70] Marco Ghiglieri. I Know What You Watched Last Sunday - A New Survey Of Privacy In HbbTV. In *Web 2.0 Security and Privacy Workshop (W2SP) 2014*, W2SP '14, 2014.

[71] Marco Ghiglieri and Michael Waidner. HbbTV Security and Privacy: Issues and Challenges. *IEEE Security & Privacy*, 14(3):61–67, 2016.

[72] Marco Ghiglieri, Melanie Volkamer, and Karen Renaud. Exploring Consumers' Attitudes of Smart TV Related Privacy Risks. In Theo Tryfonas, editor, *Human Aspects of Information Security, Privacy and Trust*, pages 656–674, Cham, 2017. Springer International Publishing.

[73] Nathan Malkin, Julia Bernd, Maritza Johnson, and Serge Egelman. "What Can't Data Be Used For?" Privacy Expectations about Smart TVs in the US. In *Proceedings of the 3rd European Workshop on Usable Security (EuroUSEC), London, UK*, 2018.

[74] Apple Inc. iTunes Preview. https://apps.apple.com/us/genre/ios/id36.

[75] Hooman Mohajeri Moghaddam, Gunes Acar, Ben Burgess, Arunesh Mathur, Danny Yuxing Huang, Nick Feamster, Edward W. Felten, Prateek Mittal, and Arvind Narayanan. ott-tracking. https://github.com/citp/ott-tracking, 2019.

[76] Apple Inc. User Interface Testing. https://developer.apple.com/library/archive/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/09-ui_testing.html, 2017.

[77] Jeff Irion. adb_shell. https://github.com/JeffLIrion/adb_shell.

[78] Google LLC. Android Debug Bridge (adb). https://developer.android.com/studio/command-line/adb.

[79] Roku, Inc. External Control Protocol (ECP). https://developer.roku.com/docs/developer-program/debugging/external-control-api.md.

[80] Lee Brotherston. TLS Fingerprinting: Smarter Defending & Stealthier Attacking. https://blog.squarelemon.com/tls-fingerprinting/, 2015.

[81] John Althouse, Jeff Atkinson, and Josh Atkins. JA3. https://github.com/salesforce/ja3.

[82] David McGrew, Brandon Enright, Blake Anderson, Lucas Messenger, Adam Weller, and Shekhar Acharya. Mercury. https://github.com/cisco/mercury.

[83] Lior Rokach and Oded Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

[84] The SciPy community. scipy.cluster.hierarchy.linkage. https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html.

[85] The SciPy community. scipy.spatial.distance.pdist. https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.pdist.html.

[86] The SciPy community. scipy.cluster.hierarchy.fcluster. https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.fcluster.html.

[87] P. Hoffman and P. McManus. DNS Queries over HTTPS (DoH). RFC 8484, RFC Editor, October 2018.

[88] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman. Specification for DNS over Transport Layer Security (TLS). RFC 7858, RFC Editor, May 2016.

[89] D. Eastlake. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066, RFC Editor, January 2011. http://www.rfc-editor.org/rfc/rfc6066.txt.

[90] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I. Hong. Why Are They Collecting My Data? Inferring the Purposes of Network Traffic in Mobile Apps. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(4), dec 2018.

[91] Konrad Kollnig, Anastasia Shuba, Reuben Binns, Max Van Kleek, and Nigel Shadbolt. Are iPhones Really Better for Privacy? Comparative Study of iOS and Android Apps, 2021.

[92] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, page 361–374, New York, NY, USA, 2016. Association for Computing Machinery.

[93] Irwin Reyes, Primal Wijesekera, Abbas Razaghpanah, Joel Reardon, Narseo Vallina-Rodriguez, Serge Egelman, Christian Kreibich, et al. "Is Our Children's Apps Learning?" Automatically Detecting COPPA Violations. In *Workshop on Technology and Consumer Protection (ConPro 2017), in conjunction with the 38th IEEE Symposium on Security and Privacy (IEEE S&P 2017)*, 2017.

[94] Jingjing Ren, Martina Lindorfer, Daniel J Dubois, Ashwin Rao, David Choffnes, and Narseo Vallina-Rodriguez. Bug Fixes, Improvements, ... and Privacy Leaks: A Longitudinal Study of PII Leaks Across Android App Versions. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, 2018.

Appendix

Appendix A supplements Section 4.1.2, Appendix B supplements Section 5, and Appendix C supplements Section 6.

A PingPong Characterization

In Section 4.1.2, we discuss how we use PingPong to extract packet pairs, which are then used as input for our clustering algorithm that analyzes the PBFs of smart TV apps. Most notably, (1) we treat app launch samples as equivalent to smart home device events: we merge the per-app launch samples into one trace and analyze it using PingPong; (2) from the output of PingPong's pair clustering step that uses DBSCAN, we make the conservative choice to only consider DBSCAN clusters that consist of identical packet pairs, i.e., we allow no variability in packet sizes in the pairs of a PBF; and (3) we use PingPong's default configuration that sets DBSCAN's ϵ parameter, which informally specifies how far a packet pair can reside from an existing DBSCAN cluster of packet pairs to become part of that cluster, to 10 [32]. However, our methodology for clustering apps based on their fingerprints is flexible enough to accommodate other design choices for the underlying packet pair extraction performed by PingPong. Next, we elaborate on the aforementioned choices (2) and (3).

(2) Packet Size Variations. In [32], PingPong is designed to allow the sizes of packets in a pair of a PLS to vary slightly. As evident from our results, in a lot of cases, smart TV apps exhibit deterministic packet pairs that always appear with identical packet sizes, e.g., C-882 S-219. However, in some cases, packet sizes can have slight variations, e.g., C-882 S-219, C-892 S-219, etc. In this paper, we make the most conservative choice and only consider packet sizes with no variations as candidates for inclusion in a PBF, discarding the remaining packet pairs that have slight variations. This choice is consistent with DBFs: Fully Qualified Domain Names (FQDN) considered as candidates for inclusion in a DBF are unique.

By loosening this strict requirement, one can potentially increase the PBFs' sizes and/or PBF prevalence, but this may come at the cost of additional PBF collisions. Our approach may be generalized to accommodate packet size variations by fine-tuning ϵ and modifying the logic that decides what clusters (as output by PingPong) are considered for inclusion in a PBF. In fact, one could even consider distance of domains in DBFs by considering not only exactly matching FQDNs, but also common effective second-level domains (eSLD).

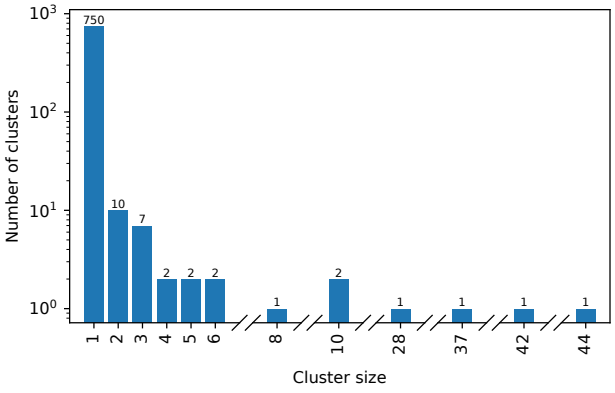

Fig. 7. Distribution of clusters by cluster size for PBFs for Roku when PingPong's ϵ parameter is set to 0.

(3) Interaction Between Pair Clustering by PingPong and PBF Clustering by FingerprinTV. Throughout this paper, we use PingPong with its default DBSCAN parameter $\epsilon = 10$ to cluster packet pairs. When we observe the output of PingPong, we sometimes find clusters that contain two different packet pairs that are within a distance of 10. For instance, we observe this phenomenon for Roku apps: the omnipresent Roku-specific packet pair described in Section 4.1.2 is sometimes clustered (by PingPong) together with other pairs that are within a distance of 10, e.g., C-882 S-219 and C-892 S-219. Since FingerprinTV discards clusters that are not exclusively comprised of identical packet pairs, it only includes the Roku-specific pair in the PBFs of 834 of the 1000 Roku apps when $\epsilon = 10$.

To eliminate this behavior, one may set $\epsilon = 0$ so as to force PingPong to only output clusters consisting of identical packet pairs. To examine what effect this has on the inclusion of the omnipresent Roku-specific packet pair in the PBFs of Roku apps, we also run PingPong with $\epsilon = 0$ and then use FingerprinTV to perform PBF extraction and performance analysis on this PingPong output. With this configuration change, the Roku-specific packet pair becomes part of the PBF for 974 Roku apps. For the remaining 26 apps, the Roku-specific packet pair appears more than L times across the L launch samples of each app and is therefore discarded (see Definition 4.2 and recall that $U = L = 10$). Compared to Figure 9c that shows 717 apps with distinct PBFs, namely apps in clusters of size 1, when we run PingPong with $\epsilon = 10$, Figure 7 shows that we have

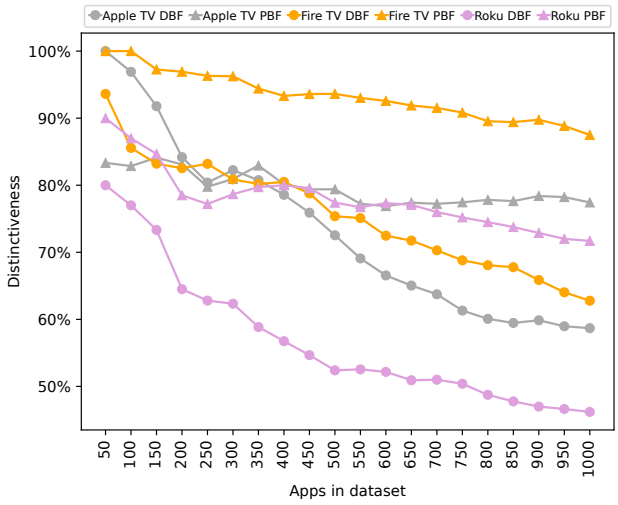

Fig. 8. Distinctiveness of DBFs and PBFs as a function of the number of apps in the dataset. Apps are added to the dataset based on the number of reviews submitted for each app. That is, $x = 50$ is a dataset comprised of the 50 most reviewed apps, $x = 100$ is a dataset comprised of the 100 most reviewed apps, and so forth.

750 apps that have distinct PBFs when we run PingPong with $\epsilon = 0$.

In summary, changing the value of ϵ in PingPong's DBSCAN clustering is a design choice. More generally, a careful co-design of DBSCAN clustering in the underlying PingPong with PBF clustering by FingerprinTV is required to achieve the desired trade-off between prevalence and distinctiveness of the extracted PBFs.

B Fingerprinting Results

This appendix includes the figures that form the basis for the results reported in Section 5.1.2 and provides further analysis of the distinctiveness of DBFs and PBFs. Figures 9a, 9b, and 9c show the number of clusters, grouped by cluster size, for PBFs for the three smart TV platforms. Figure 10 shows the distribution of PBF sizes per cluster size (the label on top of each point is the app count) for the three smart TV platforms.

B.1 Distinctiveness & Dataset Size

In Table 1, and throughout the main body of this paper, we report the distinctiveness of DBFs and PBFs for the top-1000 Apple TV, Fire TV, and Roku apps. To shed further light on the potential for fingerprint collisions,

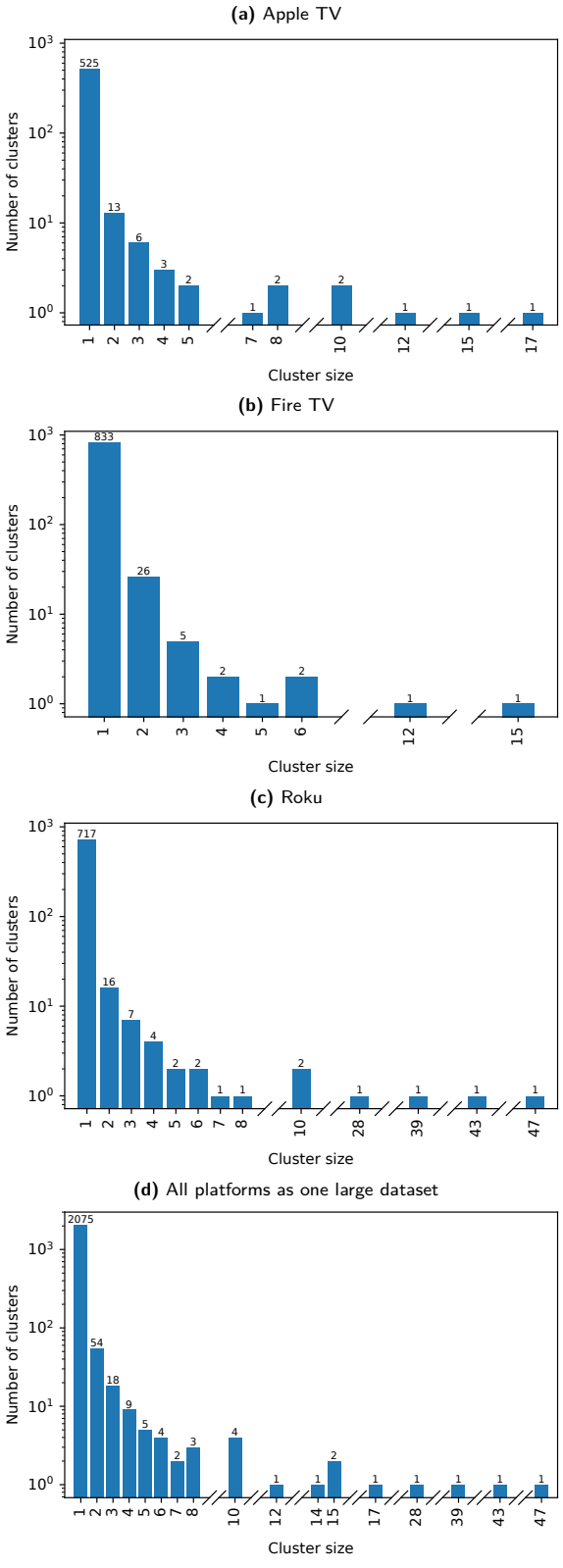

Fig. 9. Distribution of clusters by cluster size for PBFs. The cluster size is the number of apps in a cluster. For instance, the bar at $x = 2$ in Figure 9a indicates that there are 13 clusters that each contains 2 apps, for a total of 26 apps.

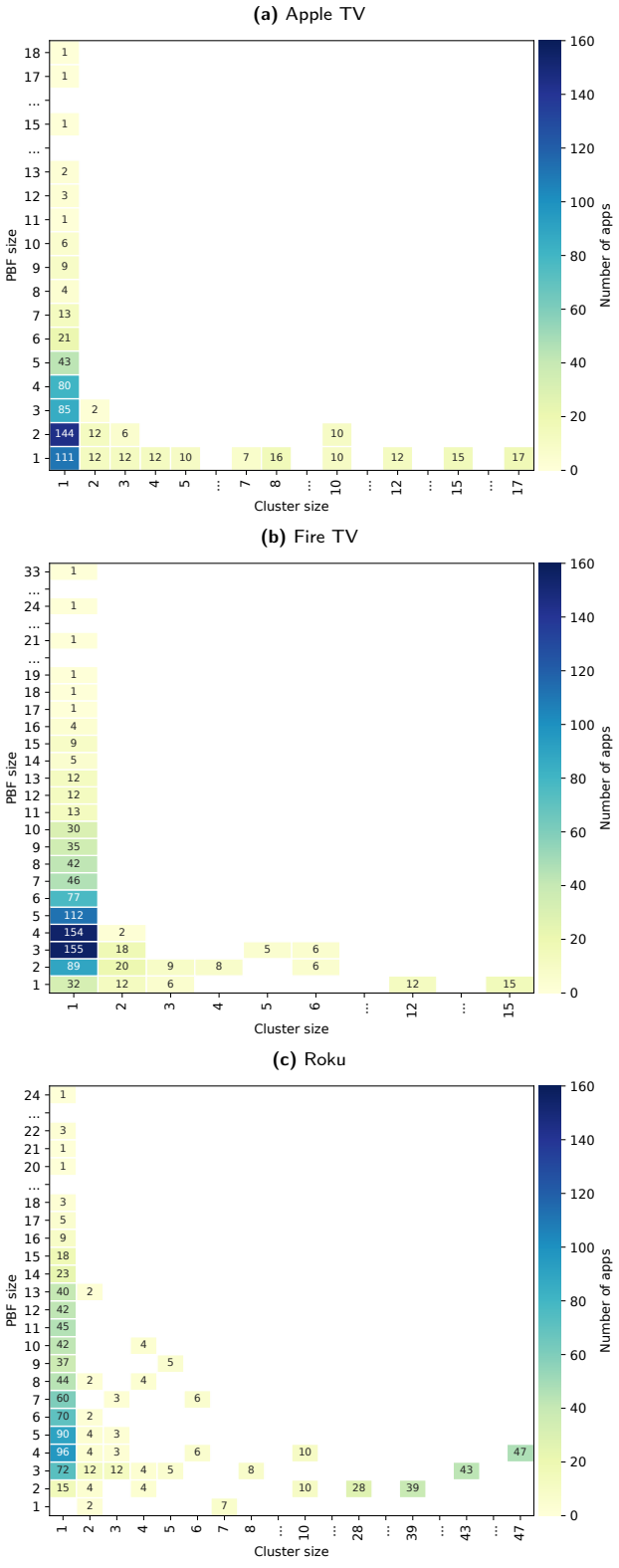

Fig. 10. Distribution of PBF sizes per cluster size. The PBF size is the number of packet pairs in a PBF. The cluster size is the number of apps in a cluster. App counts are shown for each point. For instance, the point at (2,2) in Figure 10a indicates that there are 12 apps that each has a PBF that contains 2 packet pairs and that reside in clusters that each contains 2 apps.

in Figure 8, we split each of the three datasets into increasingly larger subsets and show how the distinctiveness evolves as the number of apps considered increases. Apps are included in the subsets based on the number of user reviews submitted for each app. That is, in Figure 8, $x = 50$ is a dataset comprised of the 50 most reviewed apps, $x = 100$ is a dataset comprised of the 100 most reviewed apps, etc.

Figure 8 shows that the distinctiveness decreases slowly as the number of apps considered increases beyond 500 apps, especially for PBFs. In particular, the distinctiveness of PBFs of Apple TV apps appear to plateau. Moreover, the distinctiveness of PBFs generally declines slower than the distinctiveness of DBFs. The distinctiveness of PBFs is also generally greater than the distinctiveness of DBFs, which is in line with the observation made in Section 5.1.4.

C Mix & Match Fingerprints

This appendix includes the figures for PBFs that were omitted from Section 6.1. Figure 9d shows the number of clusters, grouped by cluster size, when the PBFs of apps of all three platforms are considered as a single, large dataset. Figure 11 shows, using one color-coded bar per platform per app, the sizes of (i.e., the number of packet pairs in) the PBFs of 60 of the 80 multi-platform apps.

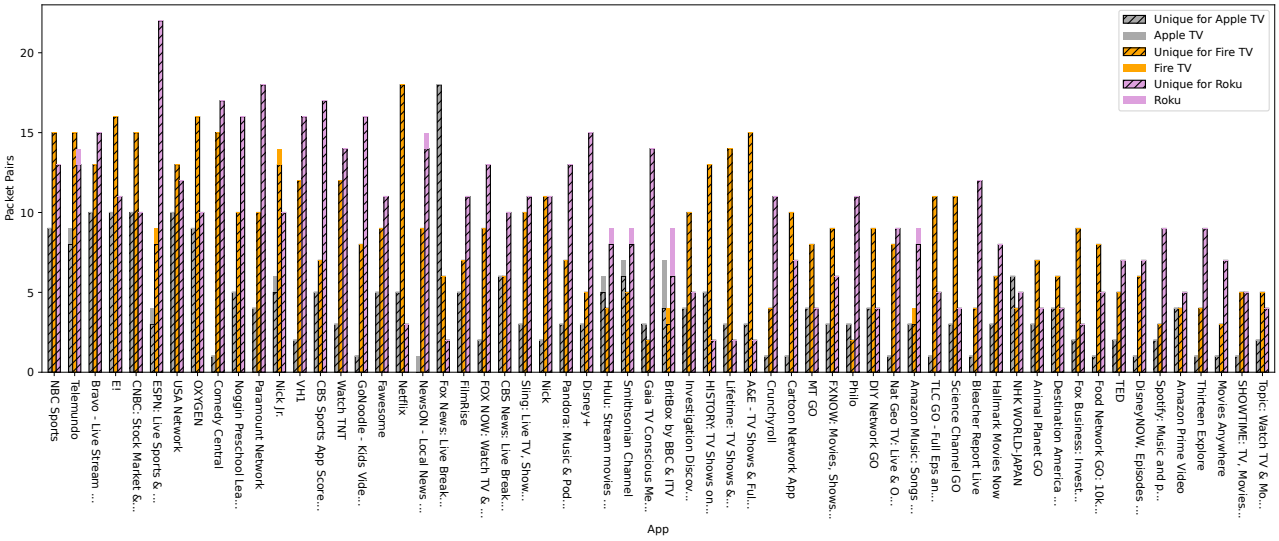

Fig. 11. PBF sizes for the 60 multi-platform apps that exhibit the largest PBFs in descending order. The size of the PBF for each version (Apple TV, Fire TV, and Roku) of an app is indicated using color-coded bars. The textured part of each bar indicates packet pairs in the PBF that are unique to the corresponding platform.