AppPrint: Automatic Fingerprinting of Mobile Applications in Network Traffic

Stanislav Miskovic[1], Gene Moo Lee[2], Yong Liao[1], and Mario Baldi[1] ⋆

[1] Narus, Inc., Sunnyvale, CA 94086
[2] University of Texas at Austin, Austin, TX 78712

Abstract. Increased adoption of mobile devices introduces a new spin to Internet: mobile apps are becoming a key source of user traffic. Surprisingly, service providers and enterprises are largely unprepared for this change as they increasingly lose understanding of their traffic and fail to persistently identify individual apps. App traffic simply appears no different than any other HTTP data exchange. This raises a number of concerns for security and network management. In this paper, we propose *AppPrint*, a system that learns fingerprints of mobile apps via comprehensive traffic observations. We show that these fingerprints identify apps even in small traffic samples where app identity cannot be explicitly revealed in any individual traffic flows. This unique AppPrint feature is crucial because explicit app identifiers are extremely scarce, leading to a very limited characterization coverage of the existing approaches. In fact, our experiments on a nationwide dataset from a major cellular provider show that AppPrint significantly outperforms any existing app identification. Moreover, the proposed system is robust to the lack of key app-identification sources, i.e., the traffic related to ads and analytic services commonly leveraged by the state-of-the-art identification methods.

1 Introduction

Mobile apps are expected to dominate Internet in the post-PC era [9]. Running on ubiquitously adopted smartphones and tablets, mobile apps support users in numerous daily activities. This attracts both individuals and enterprise users, opening a number of new opportunities. However, managing this relatively young ecosystem is still in its infancy. Even basic identification of hundreds of thousands of apps existing in Internet traffic is a challenge. This has dramatic implications on security and network management because it entails that enterprises and network operators cannot impose any meaningful policies on mobile users. Similarly, being unable to distinguish the traffic of individual apps makes isolation of malicious or infected apps difficult, if at all possible.

The key challenge we address is that mobile apps cannot be presently distinguished at many levels. First, the apps predominantly communicate with their host services via HTTP [14], which makes protocol- or port-based identification [10] ineffective. Even deep packet inspection largely fails because app traffic may not contain any *explicit app identifiers*, such as app names. In fact, we measured that these identifiers exist only in 1% of mobile traffic, while the rest is completely unknown to any characterization. Also,

⋆ Done under the Narus Fellow Research Program with equal author contributions.

a widespread use of cloud and CDN (content delivery network) services invalidates any identification based on service-host IP addresses or domain names.

In this paper, we present *AppPrint*, a system capable of identifying the apps at a granularity of arbitrarily small traffic samples. This means that app identification becomes much more frequent and its traffic coverage increases. In contrast to the state-of-the-art approaches that solely rely on temporally sparse occurrences of explicit app identifiers [2, 3, 6, 8, 11, 14], AppPrint can identify the most likely apps even when such identifiers are not present. For example, an operator of the AppPrint system could slot traffic in intervals as small as 10 seconds and identify the most likely apps for each slot and for all users in it with a high confidence. This is achievable due to two key AppPrint novelties: (i) learning elaborate app fingerprints from a priori limited app-identification data, and (ii) creating app fingerprints from features that may span multiple different traffic flows.

Specifically, we create app fingerprints as collections of *tokens*, e.g., any generic key-value pairs in URLs or any substrings of HTTP header fields, such as User-Agents or Cookies. Our intuition is that such tokens would be representative of their apps either individually or in groups; either by parameter names (keys) or parameter values. For example, an app can be designed in a specific development framework which makes the names of app parameters unique (i.e., app-identifying). Similarly, if several apps are developed on the same framework, their parameter values as well as the occurrences of specific parameters across flows could be sufficiently unique to identify each app. Leveraging this intuition enables AppPrint to characterize the traffic even when no explicit app identifiers are present. We achieve this by measuring the highest token-set similarity between an observed set of flows and readily learned app fingerprints.

One of the key contributions of AppPrint is its continual learning and refinement of app fingerprints. While the system does require some seeding with explicit app identifiers, it can expand this knowledge towards discovering many new fingerprints in the previously uncharacterised flows. Specifically, AppPrint can be bootstrapped by a single app identifier present in some particular flows (*e.g.*, embedded advertisement flows), and expand that knowledge to fingerprinting many other types of flows (such as social networking flows, audio streaming flows, etc.). We achieve this by effectively measuring collocation persistence between the seeding tokens (e.g., explicit app identifiers) and other tokens in the neighboring flows.

Another key feature of AppPrint is independence of its fingerprint learning and app identification processes. This is crucial for achieving a broad characterization coverage. Specifically, while the learning requires some seeding, the identification process has no such constraints. It can apply any fingerprints learned at any time (even offline) towards revealing the most likely apps "hidden" in the traffic. We developed an algorithm that facilitates this separation and named it MAP-SCORE.

We evaluate AppPrint on a week-long nationwide traffic trace from a major cellular provider in the United States. In this real environment, AppPrint offers an order of magnitude improvement over the state-of-the-art solutions in terms of app identification coverage.

The rest of the paper is organized as follows. Section 2 overviews the related work. Section 3 introduces AppPrint, while Section 4 evaluates the proposed methods on the

lab traffic that contains app-identification ground truth, as well as on the real ISP traffic. Section 5 summarizes this work.

2 Related Work

A number of papers have identified that network administrators increasingly lose visibility in their traffic due to mobile apps being indistinguishable from generic HTTP communications [4, 5, 13]. Hence, identifying apps via common approaches of protocol identification or port numbers is no longer effective. This situation calls for a new paradigm.

The most straightforward way to identify apps is to look for app names in HTTP User-Agent fields [14]. However, this has serious limitations on the widely accepted Android platform where app developers do not follow any conventions in creating their user agents [3], *i.e.*, the user agents may not contain app names. Another approach is to look for app identifiers in auxiliary services embedded in the apps, such as ads or analytics (A&A) [2, 3, 11]. The embedding of such services is common, especially in free apps [7, 11, 13]. However, the flow coverage of such identification is very low, because A&A flows are present in only a small fraction of traffic - especially for paid apps.

Dai *et al.* [3] improved app identification capabilities by developing a system that automatically runs Android apps and devises app fingerprints from the generated traffic. This approach can be effective, but it would be difficult to scale it to the size of current app markets with hundreds of thousands of apps. It would be even more difficult to obtain fingerprints that are representative of true human app usage, given that intelligent tools for interaction with diverse app UIs are still lacking. Thus, the obtained fingerprints may not be comprehensive or representative. For example, it would be very difficult for this system to produce representative signatures for apps that require user registration or logins, *e.g.*, the popular Android Facebook app.

Choi *et al.* [2] proposed installing a monitoring agent in mobile devices. The agent helps in building ground truth knowledge for app identification. With the data collected in a campus network, the authors generated classifiers based on HTTP user-agent fields, HTTP hostname fields, and IP subnets. However, installing such agents on user devices may be challenging due to privacy concerns. Besides, deploying the agent at a large scale has many other practical challenges.

AppPrint shares only the *initial* sources of app-identification knowledge (e.g., User-Agent fields or A&A services) with the existing approaches. These sources are only seeds that help AppPrint to learn and apply its fingerprints towards characterization of a much wider span of traffic flows.

3 Methodology

In this section, we introduce AppPrint by first providing some basic intuition behind its design. We then describe its two core algorithms: (i) *MAP*, a method for discovery and learning of new app fingerprints, and (ii) *SCORE*, a method for identification of apps in the observed traffic (based on MAP's app fingerprints).

3.1 AppPrint Overview

We are motivated by the fact that existing app identification approaches [2, 3, 6, 8, 11, 14] characterize only a small fraction of mobile traffic. AppPrint tries to increase the characterization coverage by learning a priori inconclusive features that may exist in the traffic and prove to have app identifying properties.

We focus on two types of features: (i) *tokens* that can be specific strings, or parameter names, or parameter values in HTTP headers, and (ii) *traffic flow groups* that can jointly point to an app identity. While it is a priori unknown whether useful instances of these features exist in the traffic, there is a number of reasons for them to be present. For example, app developers commonly collect statistics about their apps. Thus, there must be a way to for the apps to report back to their developers via some specific formatting of data. As an illustration, many apps use Apsalar library [1] that employs "i=" URL parameter to report executable filenames of active apps. This is an app-identifying token. Similarly, apps may require exchange of very specific parameter names to ensure proper execution. For example, Angry Birds app uses a unique "u_audio=" URL parameter to configure sound volume.

Flow grouping is another key source of AppPrint's intelligence. The grouping is invaluable when individual traffic flows do not reveal any app-specific tokens. We learned that mobile apps do exchange such generic traffic, *e.g.*, in order to transfer generic web objects such as pictures and audio. In such cases, AppPrint tries to propagate app identification from the identifiable flows to the ones that cannot be characterized. Moreover, even when none of the grouped flows contains explicit app identifiers, the tokens dispersed over several flows may jointly reveal app identity.

3.2 Initial App Identification Knowledge

Initial seeding knowledge for AppPrint can be obtained by many means. While such bootstrapping is neither the focus nor the contribution of this paper, we describe some aspects of it for completeness.

The values of explicit app identifiers are publicly available in app markets and can be easily collected for the system's bootstrap. We focus on two most popular mobile app markets: Google Play (Android) and Apple's iTunes App Store (iOS). To this end, we developed crawlers that gather the identifiers for all apps existing in these markets. For Android apps, we collect app names (such as `TuneIn Radio`) and app package denominators (such as `tunein.player`); for iOS apps, we crawled app names and unique 9-digit app IDs (such as 319295332). We verified that such tokens do appear in some URL parameters, User-Agent fields, substrings of HTTP referer fields, etc. Also, the tokens are employed by various services embedded in mobile apps, such as advertisements and analytics (A&A). An example of an app name included in the User-Agent field is illustrated in Figure 1.

3.3 MAP: Auto Fingerprint Generation

The MAP algorithm collects statistics about tokens, thus effectively serving as a knowledge repository of AppPrint. It continually self-learns and refines app fingerprints discovered in the traffic by means of flow grouping. An app fingerprint is a set of one

```
GET /Config.ashx?partnerId=xwhZkVKi&provider= ggl&latlon=
     &version=6.1&render=json HTTP/1.1
User-Agent: TuneIn Radio/6.1(Android 15;sdk;Java)
Host: opml.radiotime.com
```

Fig. 1. App identifier "TuneIn Radio" included in the HTTP User-Agent field.

or more tokens that strongly identify an app. MAP keeps a tally of all individual tokens observed in the traffic, as well as the apps to which each token may be attributed. Subsequently, if a token proves to be predominantly associated to an app, it becomes a fingerprint component of that app. Based on this principle, we next develop the MAP algorithm.

Flow grouping: *Flow grouping* is our technique to address the issue of inconclusive data, i.e., the majority of traffic that does not contain any explicit or readily known app identifiers. The idea is to first group the flows that are temporally close to each other. We perform the grouping around the instances of flows which contain known or readily learned strong app identifiers. Then, we let the flow whose app identity can be determined *suggest* or *propagate* that identity to all other tokens in its flow group - even the tokens in the neighboring (unidentifiable) flows. Consequently, in different instances of flow grouping, a token may be *suggestively* associated to different apps. To identify the most likely app, we consider all tokens and their association in the observed flows and let the SCORE algorithm decide.

The key challenge of flow grouping is that individual instances of flow groups may not be sufficient to learn true token-to-app associations. Scenarios like app multitasking and device tethering may cause flows of many apps to appear close in time, thus inducing noise in the grouping process. To eliminate such noise, one could train AppPrint in a controlled environment and ensure grouping of flows that belong to individual apps. We adopt a different strategy that is easier to scale, especially in larger networks such as cellular provider networks, university campuses or enterprise environments. We thus focus on observations of numerous flow groups generated by large user populations during AppPrint's activity. Then, given that users use different apps and run them at different times, the noise should disperse and become easily identifiable: The tokens that persistently collocate with specific apps become fingerprint tokens, while other tokens that associate equally likely with many apps get disregarded. We will verify this approach experimentally in Section 4.

An illustration of flow grouping is given in Figure 2: Flow group $g(f_x^i)$ is formed around an identifiable anchor flow f_x^i, which contains an explicit or strong app identifier. Here, parameter i designates i-th flow grouping for a given mobile user. The grouping is based on two criteria: (i) the flows must originate from the same source (source IP address), and (ii) the flows that cannot be characterized must be less than T_G seconds away from the anchoring flow f_x^i. T_G is a configurable parameter addressed in Section 4. This ensures strong time and source locality of app identification. In the illustrated example, flow f_x^i would suggest its app identity to all tokens in the group, *i.e.*, its own tokens and the tokens of other flows in $g(f_x^i)$.

MAP repository: The MAP repository is a knowledge base that reveals tokens suitable for fingerprints of each app. The repository is formed as a matrix in which each row corresponds to a token t, and each column corresponds to a *suggested* app x. The matrix

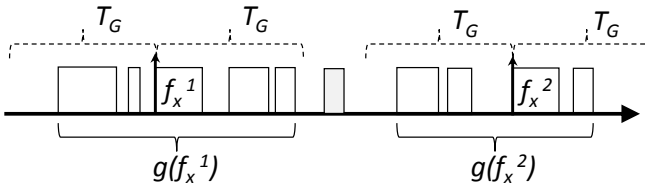

Fig. 2. MAP flow grouping.

element t, x, denoted as $MAP_{t,x}$, stores the number of instances in which token t was suggestively associated with app x.

Table 1 shows a snapshot of the MAP repository. Note the additional column in the repository which contains the total count of each token's suggested associations to any apps (denoted as column $*$ in Table 1). In the illustrated example, the repository indicates that tokens `angrybirds` and `rovio` are by far most frequently associated to *Angry Birds* app, which qualifies them for *Angry Birds* fingerprints. On the other hand, tokens such as `google` and `mobile` have dispersed associations across numerous apps, thus not being suitable for any app fingerprints.

Table 1. MAP repository example.

tokens/apps	*	*Angry Birds*	*Piggies*	*Google Maps*	...
angrybirds	500	450	0	0	...
rovio	700	600	50	0	...
mobile	3000	50	30	100	...
google	2000	60	40	200	...
...	...	...	...	...	...

3.4 SCORE: Probabilistic App Identification

SCORE algorithm determines the most likely app identities in the observed traffic. The algorithm measures similarity between the tokens found in the traffic and all token-to-app associations suggested by the MAP repository, thus identifying the most likely corresponding app. The decisions span flow sets and each decision is referred to as *app identification instance*.

SCORE flow sets: Flow sets are the units of SCORE's decision making. They are formed by bundling traffic in a different manner than MAP's flow grouping. This enables MAP and SCORE to operate independently and if needed simultaneously. The difference stems from the fact that SCORE does not need any flows with explicit app identifiers, because app identity can be readily suggested by the MAP repository. This is one of the key advantages of AppPrint: *The system is capable of identifying apps even when none of the flows (in a flow set) can be a priori characterized individually*.

For flow sets, a simple time slotting mechanism suffices: Flows that originate from the *same source* (*i.e.*, source IP address) and have starting times that fit the same time

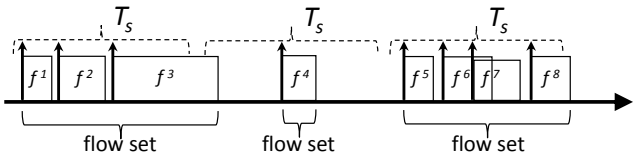

Fig. 3. SCORE grouping of flow sets.

slot constitute a flow set instance. The duration of each time slot T_S is a configurable parameter. A flow set example is illustrated in Figure 3.

SCORE and eccentricity metrics: To identify the most likely app for a given flow set, we develop a pair of metrics that leverage indications of the MAP repository. Let S_T be a set of tokens in a flow set F and S_A be the set of all apps in MAP repository. The similarity between the flow set F and an app x is evaluated as:

$$SCORE(x) = \sum_{t \in S_T} \frac{\frac{MAP_{t,x}}{MAP_{t,*}}}{\sum_{a \in S_A} \delta(t,a)}, \tag{1}$$

where $MAP_{t,x}$ is the value of the t, x element in the MAP repository, $MAP_{t,*}$ is the total number of token t's suggested associations to any apps, $\delta(t,a)$ is an indicator of t being associated to an app a in the repository, i.e., $\delta(t,a) = 1$ if $MAP_{t,a} \neq 0$.

The intuition behind the SCORE metric is the following: If tokens in the flow set S_T mostly associate with an app x, the app should score high as provided by the ratio of $MAP_{t,x}$ and $MAP_{t,*}$. Moreover, the token set should not have many other suggested app associations, as accounted by $\sum_{a \in S_A} \delta(t,a)$. The combination of these two criteria results in a high confidence that a decision about app identity for a flow set can be made unambiguously.

Once the SCORE metric for the flow set F is calculated against all candidate apps in the MAP repository, AppPrint decides whether the flow set can be attributed to the highest scoring app. This decision is based on the *eccentricity* metric. The metric requires that the score of the highest ranking app is significantly different from any other potential apps. Given the top ranked app x_{1st} and the second-best app x_{2nd}, the eccentricity ϕ is a relative difference in their scores:

$$\phi_{SCORE} = \frac{SCORE(x_{1st}) - SCORE(x_{2nd})}{SCORE(x_{1st})} \tag{2}$$

The final result positively associates app x_{1st} to the flow set if and only if the SCORE and eccentricity metrics are higher than Θ and Φ thresholds, respectively.

4 Evaluation

In this section, we evaluate the proposed MAP-SCORE algorithm (MS) against two state-of-the-art approaches for discovery of app identities: (i) one based on the content of HTTP User-Agent fields (*UA*), and (ii) another based on explicit app identifiers found in any HTTP header fields (*HH*). Both of these reference approaches rely solely on

explicit app identifiers, which makes them perfectly accurate (although on a limited set of flows).

In preparation, we conducted exhaustive sensitivity testing of the three key MAP-SCORE parameters: (1) flow (set) grouping interval T, (2) threshold Θ of the SCORE metric, (3) threshold Φ of the SCORE eccentricity. We found that app classification is largely consistent over various parameter settings whenever T is around 10 seconds, Θ is between 0.1 and 0.2 and Φ is around 0.3. The parameters set for our experiments are $T = 10s$, $\Theta = 0.1$ and $\Phi = 0.3$.

4.1 Datasets

Lab Trace: To evaluate AppPrint, we partly use lab traffic generated by running individual apps in order to establish a ground truth. To this end, we downloaded $40K$ Android apps from Google's Play Store and collected their traffic. Each app was run on multiple versions of Android emulators provided by the Android SDK. We use the Android monkey tool [12] to emulate user interaction with the apps. Similarly, we collected $7K$ popular apps from Apple's iTunes App Store. Given that Apple does not provide any emulators for iOS devices, we developed one and enabled it to automatically install and execute apps, as well as collect app traffic.
Real Trace: We also evaluate AppPrint on a large anonymized dataset from a major US cellular provider. The dataset contains 7 days of traffic from about $200K$ anonymous and mostly Android users. This dataset faithfully represents actual human usage of mobile apps. However, it does not provide any a priori information of apps behind the traffic, except for a small portion of flows (less than 1%) whose apps can be determined via User-Agent (UA) or header data (HH) approaches.

4.2 App Identification

We first evaluate the traffic coverage characterized by MAP-SCORE (MS) in the real trace. Due to the lack of comprehensive ground truth in this trace, it is impossible to fully evaluate correctness of MAP-SCORE's results. Thus, we later conduct precision analysis on the lab trace in order to provide a holistic view in AppPrint's capabilities.

Our experiments use the first 6 days of the real trace to provide training for the MAP repository. Our evaluation is based on running SCORE against the flow sets in the 7^{th} day of data. The same evaluation methodology is used for the other two approaches, user agents (UA) and header data (HH), i.e., we evaluate their characterization capabilities only on the 7^{th} day of data.

Coverage of the real traffic: The number of *identified app instances* is used as a coverage comparison metric. For MAP-SCORE, an app-instance identification corresponds to SCORE positively associating an app fingerprint to a flow set. By design, there can be at most one such app identification per flow set. For HH and UA, we count the total number of *distinct* app identifications in each flow set - i.e., depending on the number of different explicit app identifiers found in the flow set, there may be more than one app identified per flow set.

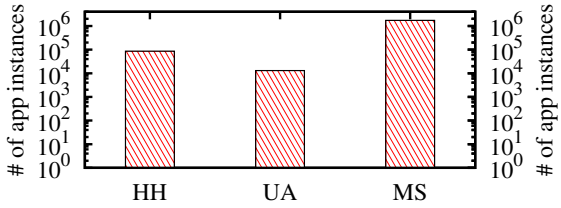

Fig. 4. Number of identified app instances.

As plotted in Figure 4, MAP-SCORE (MS) identifies $1,729K$ app instances, while UA identifies close to $13K$ app instances and HH about $86K$ app instances. Coverage-wise, MAP-SCORE performs an order of magnitude better. We also note that UA is not as effective as described in [14]. This is due to the fact that "our" cellular provider mainly supports Android, the platform that doesn't force developers to code explicit app identifies in User-Agents. In contrast, the trace studied in [14] included a significant portion of the traffic from iOS devices, whose apps predominantly include app-identifying information in User-Agents.

Among $1,729K$ positive fingerprint matches of MAP-SCORE, we found that $84K$ are consistent with the indications of UA or HH approaches. This can be used as a hint of MAP-SCORE's accuracy. Further accuracy analysis could not be conducted on the real trace because it does not contain the ground truth for the remaining $1645K = 1729K - 84K$ MAP-SCORE app identifications.

Precision evaluation: In order to further assess accuracy of MAP-SCORE results, we use the lab trace which does contain the ground truth of flow-to-app associations. We built the trace by combining lab-generated traffic of 1000+ apps which appeared in the MAP repository (i.e., the repository trained on the first 6 days of the real trace). Then, SCORE was run against such traffic. To evaluate app-identification precision and coverage, we use the standard ratios of true positive and false positive detections. Our results indicate that MAP-SCORE achieves 81% flow-set coverage with 93.7% precision. This supports our claim that AppPrint can identify the most likely apps with a wide flow-set coverage and with a high confidence.

Further, this result supports our assumption about the noise canceling properties of MAP-SCORE fingerprints (stated in Section 3.3). Specifically, even though MAP was trained on the noisy real traffic, fingerprint indications were still highly precise when applied on the ground truth of the lab traffic. Thus, the app fingerprinting noise largely dispersed over time and large user population as we expected.

4.3 Effectiveness of Grouping Flows

Next, we evaluate the importance of flow grouping, *i.e.*, the importance of using tokens from multiple flows towards building app fingerprints. We take an extreme approach by preventing any grouping, i.e., we apply AppPrint on the flow sets containing only single flows. The settings for these experiments are similar to the ones described in Section 4.2.

Real traffic: We first evaluate the number of identified app instances on the 7^{th} day of the real traffic. The results for header data (HH) and user agent (UA) approaches do not change significantly; a slight increase in identification is due to counting distinct

app identifications per each flow vs. counting them once per flow set. Next, this setting enables us to better qualify MAP-SCORE (MS) results. Specifically, using per-flow indications of deterministic HH and UA approaches, we can classify MS results as (i) true positives (TP) when MS agrees with HH or UA, (ii) false positives (FP) when there is a disagreement, and (iii) non-verifiable characterizations (NV) when HH or UA cannot characterize a flow, but MS can.

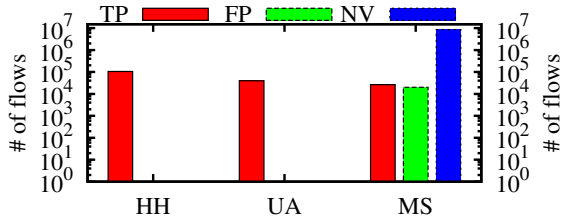

Fig. 5. Number of identified app instances without flow grouping.

The results in Figure 5 indicate that although MS identifies one order of magnitude more app instances than with flow grouping enabled (compare Figure 5 and Figure 4), the number of true positives decreases from $85K$ to $26K$. This clearly demonstrates a positive effect of flow grouping on AppPrint's accuracy.

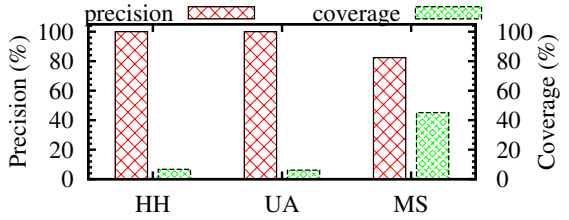

Fig. 6. Precision and coverage of app-instance identifications in the lab trace without flow grouping.

Lab traffic: We use the lab trace to assess the impact of single-flow flow sets on precision (see Figure 6). Our results indicate that MAP-SCORE (MS) achieves much higher coverage than HH and UA, but its precision drops to 82.3% (about 10% less than with flow grouping enabled, see Section 4.2). In summary, not leveraging tokens from multiple flows has notable negative effects on both coverage and precision of AppPrint.

4.4 Identifying Apps without A&A Traffic

To evaluate AppPrint's capabilities on paid apps (without incurring high monetary costs of purchasing the apps), we leverage the key difference between free and paid apps: Paid apps mostly do not exchange ads and analytic (A&A) traffic [7, 13], while the rest of their communications are largely similar to free apps. Thus, we can still employ our lab- and real-traffic traces by removing all A&A flows. Also note that our MAP repository still remains representative because most paid apps have their free counterparts developed on the same code base, thus using similar traffic tokens (readily captured by our

MAP repository). Our experimental settings are otherwise similar to the ones described in Section 4.2.

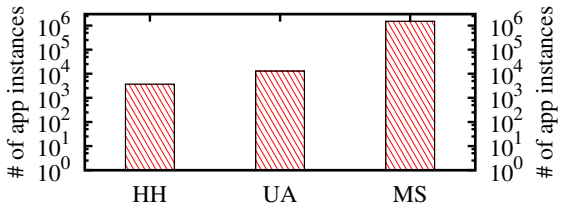

Fig. 7. Number of app instances identified in the real trace without A&A flows.

Figure 7 shows the number of app instances identified by the three techniques on the real trace. In this experiment header data (HH) approach identifies only $3.7K$ app instances, user agent (UA) approach identifies $13K$ (remaining unaffected), and MAP-SCORE (MS) identifies $1,508K$. In comparison with A&A flows included (see Figure 4), MAP-SCORE identifies only about 11% less app instances.

For the lab trace, precision and coverage are plotted in Figure 8. Note that the coverage of HH gets very low because this technique largely relies on A&A traffic for explicit app identifiers. UA still has a low coverage, but it is largely unaffected by the lack of A&A flows. Finally, MAP-SCORE associates apps to about 50% of flow sets, but its precision drops to around 70%.

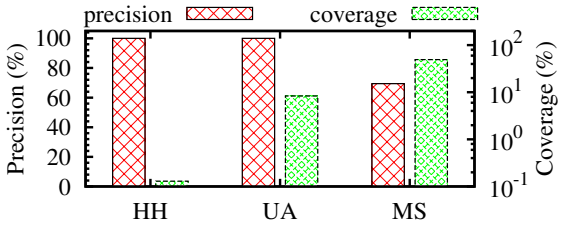

Fig. 8. Precision and coverage of app-instance identifications on the lab trace without A&A flows.

5 Conclusion

The paper proposes AppPrint, a system for automatic identification of mobile apps in arbitrarily small samples of Internet traffic. AppPrint enables network administrators to regain fine grained visibility into their traffic, thus benefiting network management and security. The system achieves this by its unique capability to learn app fingerprints dispersed over multiple and often individually inconclusive traffic flows. We evaluated AppPrint on a trace of a large cellular provider in the United States and on our comprehensive lab trace spanning thousands of apps. The results show that AppPrint outperforms state-of-the-art approaches by identifying over one order of magnitude more instances of apps in the real traffic, while achieving up to 93.7% precision.

References

1. Apsalar: Data-Powered Mobile Advertising, http://apsalar.com/
2. Choi, Y., Chung, J.Y., Park, B., Hong, J.W.K.: Automated classifier generation for application-level mobile traffic identification. In: Proceedings of Network Operations and Management Symposium (NOMS) (2012)
3. Dai, S., Tongaonkar, A., Wang, X., Nucci, A., Song, D.: NetworkProfiler: Towards automatic fingerprinting of Android apps. In: INFOCOM. Turin, Italy (Apr 2013)
4. Falaki, H., Lymberopoulos, D., Mahajan, R., Kandula, S., Estrin, D.: A first look at traffic on smartphones. In: Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement. pp. 281–287. IMC '10, ACM, New York, NY, USA (2010)
5. Falaki, H., Mahajan, R., Kandula, S., Lymberopoulos, D., Govindan, R., Estrin, D.: Diversity in smartphone usage. In: Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services. pp. 179–194. MobiSys '10, ACM, New York, NY, USA (2010)
6. Gember, A., Anand, A., Akella, A.: A comparative study of handheld and non-handheld traffic in campus Wi-Fi networks. In: Proceedings of the 12th International Conference on Passive and Active Measurement. PAM'11, Berlin, Heidelberg (2011)
7. Leontiadis, I., Efstratiou, C., Picone, M., Mascolo, C.: Don't kill my ads!: Balancing privacy in an ad-supported mobile application market. In: Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications. pp. 2:1–2:6. HotMobile '12, ACM, New York, NY, USA (2012)
8. Maier, G., Schneider, F., Feldmann, A.: A First Look at Mobile Hand-held Device Traffic. In: Proceedings of the 11th International Conference on Passive and Active Measurement. pp. 161–170. PAM'10, Springer-Verlag, Berlin, Heidelberg (2010)
9. Mobile App Usage Further Dominates Web, http://www.flurry.com/bid/80241/Mobile-App-Usage-Further-Dominates-Web-Spurred-by-Facebook#.VAZhp9-c3PE
10. Moore, D., Keys, K., Koga, R., Lagache, E., Claffy, K.C.: The coralreef software suite as a tool for system and network administrators. In: Proceedings of the 15th USENIX Conference on System Administration. pp. 133–144. LISA '01, USENIX Association, Berkeley, CA, USA (2001)
11. Rastogi, V., Chen, Y., Enck, W.: AppsPlayground: automatic security analysis of smartphone applications. In: Proceedings of the Third ACM Conference on Data and Application Security and Privacy. pp. 209–220. CODASPY '13 (2013)
12. UI/Application Exerciser Monkey, http://developer.android.com/tools/help/monkey.html
13. Wei, X., Gomez, L., Neamtiu, I., Faloutsos, M.: ProfileDroid: Multi-layer Profiling of Android Applications. In: Proceedings of the 18th Annual International Conference on Mobile Computing and Networking. pp. 137–148. Mobicom '12, ACM, New York, NY, USA (2012)
14. Xu, Q., Erman, J., Gerber, A., Mao, Z., Pang, J., Venkataraman, S.: Identifying Diverse Usage Behaviors of Smartphone Apps. In: Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference. pp. 329–344. IMC '11, ACM, New York, NY, USA (2011)