Mobile-APP Fingerprints on Encrypted Network

Yifei Pang
yifeip@andrew.cmu.edu

Zhiyue Lyu
zhiyuely@andrew.cmu.edu

Abstract

FLOWPRINT is a semi-supervised approach for fingerprinting mobile apps from (encrypted) network traffic. We base on it, improves the model by removing DNS traffic which does little help in generating app specific fingerprints. The new model proves to outperform the old model both on original app traffic dataset and the new dataset in 2021. In the evaluation of the new dataset, we achieved an accuracy of 85.77% for recognizing apps compared with 85.54% of the old model, and a precision of 98.86% for detecting unseen apps, compared with 86.42% of the old model.

As the extension, we also experiment the new model on websites recognition and detection in browser traffic. The traffic with a single website running in a browser tends to behave well while multiple simultaneous websites will be more difficult to deal with. We proposes two possible ideas to improve the model and its application on browser traffic at the end of the paper.

Keywords: mobile-app fingerprints, encrypted network, machine learning, browser traffic

1 Introduction

In the evolving landscape of mobile technology, the ability to identify and monitor mobile applications through network traffic has become a critical component of network security. The identification of running applications was straightforward before, as network traffic just contained plaintext data without encryption. However, with the development and widespread deployment of encryption protocols such as SSL/TLS, traditional methods seem no longer feasible.

This challenge has inspired new approaches in the field of mobile apps identification, one of which is the semi-supervised machine learning model FLOWPRINT[1]. FLOWPRINT offers a novel way to generate and identify fingerprints of mobile apps through encrypted network traffic. The model extracts unique traffic patterns from network flows, which mainly focus on destination IP addresses and port, and TLS certificate, as well as their timestamps.

As mobile apps are now more complicated and the network traffic becomes more intense, we have doubt about the effectiveness and scalability of FLOWPRINT. Our project builds upon this foundation with the goal of understanding, evaluating, and extending FLOWPRINT's capabilities. We aim to not only assess the model's current efficiency but also to enhance its performance through modifications using more recent app traffic data. Additionally, we explore the possibilities of applying FLOWPRINT model to broader regions such as browser traffic.

2 related work

Flowprint is a semi-surprised machine learning algorithm for apps fingerprints generation and comparison. It bases on destination tuple of IP and port, and TLS certificate to cluster flows to many destinations, then use their co-occurrence frequency to extract a highly correlated destinations as a fingerprint. Later, these fingerprints can be used to compare and decided as one of the know app or unseen app. The model enables fast generation of fingerprints compared with other methods. It can also recognize known apps and detect unseen apps with only encrypted traffic flows after training. Therefore, it is a descent option for apps monitoring and malware detection and recognition.

There are many other research also explored the use of network fingerprints for mobile apps. AppScanner [2] uses statistical features of packet sizes in TCP streams to train Support Vector and Random Forest Classifiers for recognizing known apps, which achieves high accuracy in the top 110 most popular apps in the Google Play Store apps. However, it has difficulty to classify for all TCP flows. NetLang [3] presents a method that combines machine learning and automata learning to create a sophisticated system for classifying network traffic, while it is rather slow. Later, ML-NetLang [4] improves NetLang [3] and FLOWPRINT [1] by using automata learning algorithm to observe the temporal order among destination-related features of network traffic and create a language as a fingerprint.

While in this paper, we did not delve deeper into the state-of-art fingerprint technique. Instead, we still work on FLOWPRINT, and try to understand the principle of it. We believe research on it may do help to the related research in the future because these simple features have quite big mutual information, so it potentially offers novel insights and valuable methodologies for further studies, providing a foundation for advancements in the field of fingerprint techniques.

3 project goals

This section outlines the specific goals that we aim to accomplish in our project. Each one is designed to progressively understand, improve and extend FLOWPRINT:

1. **Implementation and usage**

   The first goal is to effectively implement and utilize FLOWPRINT. This involves setting up necessary environment and exploring how to modify the model.

2. **Evaluation with original dataset**

   Our second goal is to evaluate FLOWPRINT using the dataset originally used by its own paper[1]. This evaluation serves as a baseline to measure the model's efficiency on recognizing known apps and detecting unseen apps. Through this, we can have a better understanding of the model's principles.

3. **Examine the mismatches and improve the model**

   One of our key goals is to explore the reason for mismatches during the initial evaluations with original dataset. By analyzing these mismatches, we find that different apps' fingerprints may contain some common features generated from public service providers. We aim to improve the model to reduce these common features to improve the performance.

4. **Collect more recent datasets for analysis**

   Given the rapid evolution of mobile apps and encryption methods, the original datasets in and before 2016 may be obsolete. We try to collect more recent data for evaluation, while most of them comes from Mappgraph[5] with data collected in 2021.

5. **Random search on hyperparameters to evaluate**

   To optimize the new model's performance, we implement a random search on its hyperparameters, including window size, correlation, and similarity. We assess the new model under different groups of hyperparameters, to find the best one on the new model.

6. **Extend application on browser**

   Finally, we explores the possibilities of applying the new model to browser traffic. We try to identify the websites running on the browser. This extension aims to test the model's capabilities in more complex scenarios, such as the traffic of a browser with a single website and with multiple websites.

4 Preliminary analysis

In the initial part of our project, we conducted a thorough analysis of the original FLOWPRINT model to understand its capabilities and limitations. This analysis served as a foundation for our subsequent efforts to enhance the model.

4.1 Capabilities on app recognition

The original FLOWPRINT model employs a semi-supervised learning approach to generate and match mobile apps fingerprints. By extracting features such as destination IP with port and TLS certificate serial number, then correlating them across time windows, the model could identify applications even from the encrypted data.

As is shown in Figure 1, for app recognition, each part comes with true label on the first line. Then comes several fingerprints with predicted labels.

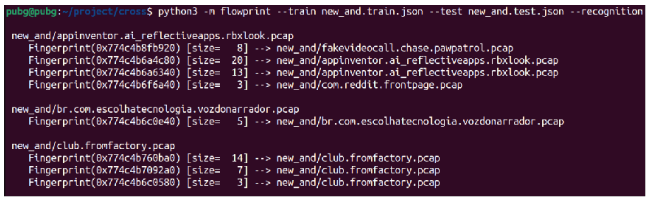

Figure 1: The capability on APP recognition of the old model

This capability underscores the model's utility in conditions where mobile application recognition is critical for network management and security, such as detecting malicious apps.

4.2 Capabilities on unseen app detection

FLOWPRINT also exhibits capabilities in detecting unseen applications, which is not part of the training dataset. The model's design allows it to flag new app fingerprints that do not match any previously identified patterns by a threshold.

The following Figure 2 demonstrates that for unseen app detection, FLOWPRINT returns "1" for each flow matching a known app or a "-1" for apps that are not recognized, indicating these apps are unseen.

This feature is useful for achieving security network environments when new applications are introduced, which helps identify potentially malicious unknown apps running over the network.

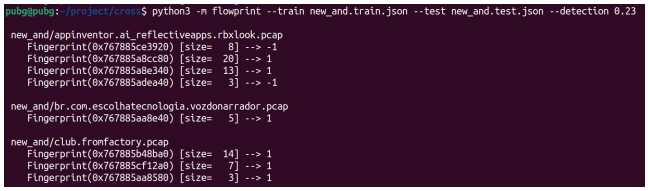

Figure 2: The capability on APP detection of the old model

4.3 Mismatches

Despite the model's original strengths on network management and security, it still faces challenges with mismatch problems.

From Figure 1, we could see the model makes false prediction for the first app, matching *fakevideocall* to *appinventor*. Moreover, from Figure 2, it shows a misprediction for a known app, which should output "1" instead.

By checking the fingerprint file in detail, Figure 3 shows that although the two apps are different, they share the same destination to a public DNS resolver.

Figure 3: Mismatch analysis

Since these public DNS resolvers are widely used across applications, these services often lead to non-specific fingerprints that could obscure the model's identification, thus reducing the performance of FLOWPRINT.

As a result, the issue above highlights the need for model improvements, such as stripping off DNS flows to refine the fingerprinting process.

5 Model Implementation

In this section, we describe the logic structure of the old model and our advanced new model.

5.1 Old Model

The old model of FLOWPRINT employs a series of steps to fingerprint mobile applications based on encrypted network traffic. The basic steps are shown as a flow graph in Figure 4. Here is a detailed breakdown:

5.1.1 Feature Extraction

In this initial stage, FLOWPRINT extracts critical network features that remain observable even within encrypted traffic:

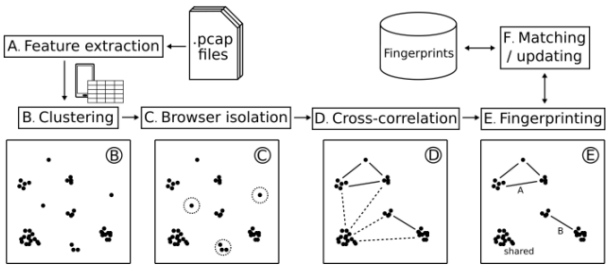

Figure 4: Basic steps of FLOWPRINT[1]

- **Destination IP and Port:** The tuple of destination IP and port offers a basic but crucial understanding of the network endpoints that different applications communicate with.

- **TLS Certificate Serial Number:** This feature exploits the visibility of TLS certificate details. By analyzing the serial number of TLS certificates, FLOWPRINT can distinguish between different encrypted flows, tying them back to specific services or entities.

- **timestamp:** The timestamp of the flow is a significant information to correlate flows. Those traffic usually in the same window unit, such as 5 second, are probably from the same apps. We calculate their simultaneous frequency to get their correlation-ship, which then help generate fingerprints with highly correlated flow features.

5.1.2 Flow Clustering

Flow clustering leverages similarity metrics to group network flows, enhancing the identification of app-related patterns:

- **Clustering Criteria:** Flows are clustered based on shared destination tuples or TLS certificates. This grouping is essential to reduce the amount of data and aggregate the same flows.

5.1.3 Browser Isolation

The model isolates browser-generated traffic to prevent the masking of characteristic app traffic patterns:

- **Isolation Technique:** Special algorithms identify and segregate browser traffic, which is characterized by high destination diversity and non-specific access patterns. This step is vital for ensuring that the data used in further analysis is indicative of dedicated app behaviors rather than generic browsing activity.

- **Impact:** Isolating browsers significantly refines the dataset, allowing for more accurate fingerprinting of actual app traffic by removing highly variable and non-indicative traffic data.

5.1.4 Correlation Discovery

This stage involves identifying temporal and behavioral correlations between the clustered flows:

- **Correlation Analysis:** By examining the timing and co-occurrence frequencies of network access to various destinations, FLOWPRINT identifies patterns that are likely to represent the network behavior of specific apps.

- **Tools and Techniques:** Utilization of statistical correlation techniques such as Pearson correlation coefficients to quantify the strength of relationships between different network destinations.

5.1.5 Fingerprint Creation

Utilizing identified correlations, FLOWPRINT constructs unique fingerprints for each app:

- **Fingerprint Definition:** A fingerprint comprises a set of network destinations that frequently co-occur for an application, effectively capturing its typical network behavior.

- **Construction Method:** Fingerprints are extracted as maximal cliques from highly correlated clusters over a threshold, where high correlation indicates a higher likelihood of originating from the same app.

5.1.6 Matching and Detection

In the final implementation phase, the model matches new network fingerprints against these fingerprints to recognize and categorize apps:

- **Matching Algorithm:** Utilizes a similarity assessment tool, such as the Jaccard similarity, to evaluate how closely incoming traffic matches the established fingerprints. The Jaccard similarity is given by:

$$J(F_a, F_b) = \frac{|F_a \cap F_b|}{|F_a \cup F_b|}$$

- **Detection Capability:** The system is capable of recognizing known apps and detecting anomalies or new app behaviors by identifying traffic that does not correspond to any existing fingerprint.

5.2 Advanced New Model

The refined new model introduces a significant enhancement to the app fingerprinting process by addressing the issue of DNS-related traffic. Recognizing that flows to common DNS resolvers can obfuscate app-specific patterns, the model strategically excludes such traffic to improve the precision of its analysis.

5.2.1 Rationale for Excluding DNS Resolvers

DNS traffic to common resolvers, such as Google Public DNS at 8.8.8.8 and 8.8.4.4, Cloudflare at 1.1.1.1 and 1.0.0.1, Quad9 at 9.9.9.9 and 149.112.112.112, and OpenDNS at 208.67.222.222 and 208.67.220.220, are ubiquitous and therefore offers little app-specific insights. These resolvers improve speed, reliability, enhanced security, and privacy for users, but do not contribute to app-specific fingerprinting in network traffic analysis. Consequently, the new model excludes these flows to refine the fingerprinting process.

5.2.2 Implementation Code Snippet

The following code snippet outlines the process of removing DNS-related traffic:

```
for packet in traffic:
    if '53' in packet:
        continue
```

This conditional statement checks if the string "53" is present in the packet representation, which suggests that the packet is related to DNS traffic. If port "53" is detected, the *continue* statement is executed, which skips the current iteration of the loop and proceeds to the next packet. The new model ensures that any packet involved in DNS operations is not included in subsequent analysis stages. This strategy specifically tailors the dataset for app-specific traffic evaluation by retaining only those packets that are more likely to carry app-generated data, thereby enhancing the model's focus and efficiency in characterizing application behavior based on network traffic.

6 Outcomes and Results

In this section, we evaluate the performance of both the old and the advanced new model, focusing on their effectiveness in accurately fingerprinting mobile applications over encrypted network traffic. We present a correlation graph to visualize network flow connections, evaluate the two models' performance using the original dataset, examine the impact of hyperparameter adjustments on new dataset, and compare the old and new models on the best hyperparameter.

6.1 Correlation Graph Analysis

The correlation graph is a visual representation of network traffic, where node sizes correspond to the frequency of destination visits, and edge thickness indicates the strength of the correlation between destinations. By identifying highly correlated nodes, the model deduces significant app-related traffic patterns, crucial for fingerprint creation.

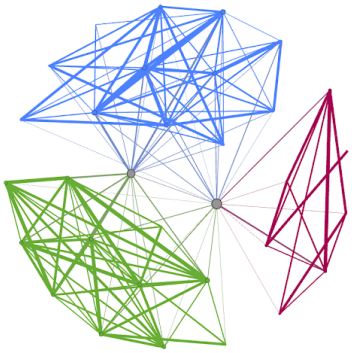

Figure 5: **Visualization of a correlation graph for network traffic analysis, categorizing flows from different services**. Clusters represent *Amazon* in green, *Sephora* in blue, *Indeed* in red, and common or public destination in grey.

Figure 5 shows the correlation graph of an extracted flow example. Clusters corresponding to the same application exhibit significant cross-correlation. Conversely, shared clusters exhibit only minimal correlations across different applications, and the majority of distinct clusters display no notable correlation with one another.

6.2 The performance of the models on original data

The original dataset is used to assess the model's accuracy It comprises traffic data from a diverse set of mobile applications across various platforms. It includes packets from 100 applications each from the US (e.g., Amazon, Tinder, Steam), China (e.g., Tencent, Sina, Qzone), and India (e.g., Cybrook, ShopClues, ixigo) under the **Cross Platform** dataset. Additionally, it features data from 1.03 million Android applications included in the **Andrubis** dataset, which gathers applications from Google Play Store and 15 alternative markets. Each packet from these datasets contains 1 to 20 fingerprints.

We employ an automatic shell script to randomly select 100 packets from each dataset to ensure a randomized and unbiased selection process for the evaluation. This method helps in accurately determining the model's effectiveness in recognizing applications it has been trained on, as well as its capability to detect new, unseen applications. The diversity and size of the datasets provide a robust foundation for testing the model under realistic and varied conditions.

6.2.1 App Recognition

For app recognition analysis, the training dataset is composed of the training part of packets from both Cross Platform and Andrubis datasets, while the test dataset comprises only the test part of packets from the Cross Platform dataset.

The results of the original and new models on these datasets are summarized in Table 1:

Metric	Original Model	New Model
Fingerprints	500	453
Accuracy	0.8780	0.9426
Precision	0.8953	0.9358
Recall	0.8018	0.9007
F1 Score	0.8460	0.9179

Table 1: Comparison of performance metrics between the original and new models for app recognition

These results highlight the improvements in recognition accuracy, precision, recall, and F1 score achieved by the new model compared to the original model, illustrating its enhanced ability to effectively process and recognize the diverse application traffic within the original datasets.

6.2.2 Unseen App Detection

For the detection tasks, the dataset was partitioned into a training set comprising the training part of packets from the Cross Platform dataset and a test set consisting of the test part of packets from both the Cross Platform and Andrubis datasets. This configuration aims to evaluate the model's robustness across a more extensive and diverse test scenario, enhancing the reliability of the detection outcomes.

The performance metrics of the original and new models on the detection task are presented in Table 2:

Metric	Original Model	New Model
Fingerprints	546	546
Accuracy	1.0000	0.9927
Precision	1.0000	0.9856
Recall	1.0000	1.0000
F1 Score	1.0000	0.9927

Table 2: Performance comparison of original and new models for detection tasks

The results highlight the original model's exceptional performance, achieving perfect scores across all metrics. The new model, while slightly lower in accuracy and precision, maintains a perfect recall rate, demonstrating its high effectiveness in identifying targeted packets within the datasets.

6.3 Hyperparameter Optimization

To evaluate the performance impact of different hyperparameters on our model, we utilized datasets from two distinct sources: **MAppGraph** and **Cross-Platform**. The new introduced MAppGraph dataset includes packets from 30 applications known for their complex and variable network behavior, such as Telegram, Messenger, and Among Us, with each packet containing between 20 to 100 fingerprints.

This diverse collection of applications and packet sizes enables a comprehensive assessment of our model's ability to adapt and perform under various conditions influenced by hyperparameter changes. By altering parameters such as window size, and observing the corresponding changes in model performance metrics like accuracy, precision, and recall, we aim to identify the optimal settings that yield the best overall performance for application recognition and traffic detection tasks.

We implement a random search strategy to identify the most effective combination of hyperparameters that influence model performance. The table below summarizes 20 hyperparameters groups we test, including batch size, window size, correlation, and similarity.

Batch	Window Size	Correlation	Similarity
300	10	0.08	0.67
300	10	0.09	0.86
300	10	0.15	0.81
300	10	0.17	0.63
300	10	0.20	0.71
300	10	0.24	0.51
300	10	0.39	0.66
300	10	0.43	0.86
300	20	0.12	0.66
300	20	0.18	0.85
300	20	0.29	0.63
300	20	0.29	0.86
300	20	0.39	0.64
300	30	0.10	0.90
300	30	0.11	0.52
300	30	0.21	0.65
300	30	0.28	0.50
300	30	0.28	0.72
300	30	0.33	0.81
300	30	0.36	0.66

Table 3: Hyperparameter configurations tested during the random search

6.3.1 App Recognition

The datasets used in the recognition task are partitioned as follows:

- **Training Dataset**: Consists of the training portion of packets from *MAppGraph* and additional packets from *Cross Platform* sources.
- **Test Dataset**: Comprises the test part of packets from *MAppGraph*.

The performance of the app recognition is assessed based on different hyperparameters, with the primary metric being the F1 Score. The scatter plot in Figure 6 illustrates the F1 Scores achieved with varying similarity and correlation hyperparameters, using different window sizes as markers. Higher F1 Scores, indicated by a gradient from blue to red, suggest better recognition performance.

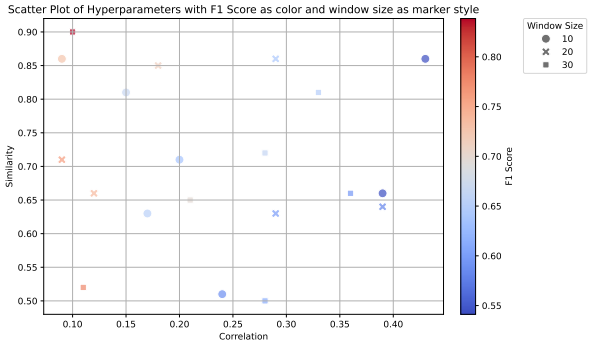

Figure 6: Scatter plot of hyperparameters with F1 Score on app recognition

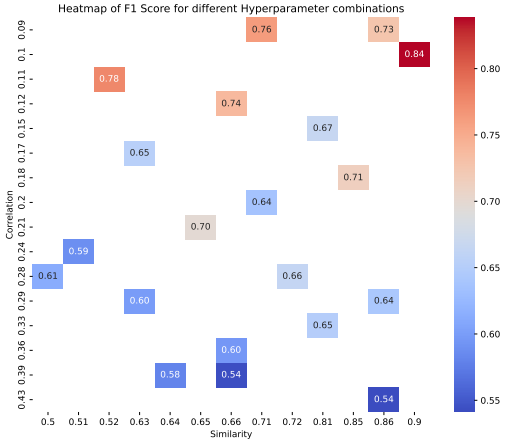

Figure 7: Heatmap of F1 Score on app recognition

The heatmap presented in Figure 7 illustrates the F1 Scores in detail. The x-axis of the heatmap represents the similarity values ranging from 0.5 to 0.9, while the y-axis displays correlation values from 0.09 to 0.43. The highest F1 Scores,

as indicated by the darkest red colors, are predominantly observed with correlation set to 0.1, similarity set to 0.9 and window size set to 30. This pattern suggests that the closer the hyperparameters to the standard above, the test data leads to more accurate recognition.

For further verification, we utilize window size as a baseline to conduct further tests on correlation and similarity. The following graphs(Figure 8, 9, 10 confirms our previous conclusions.

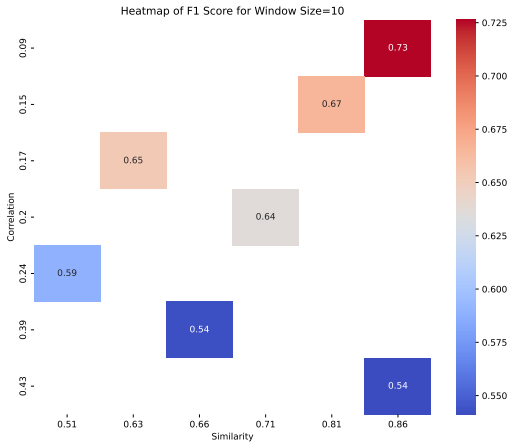

Figure 8: Heatmap of F1 Score with window size = 10 on app recognition

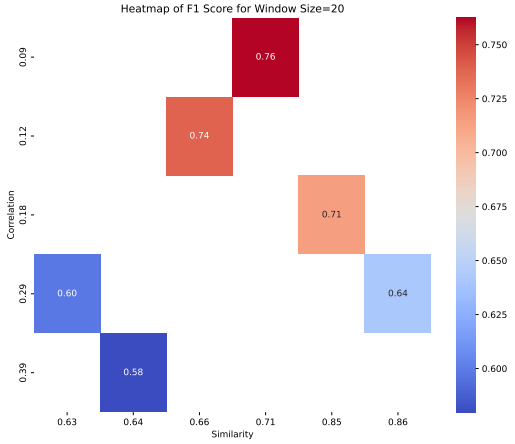

Figure 9: Heatmap of F1 Score on with window size = 20 on app recognition

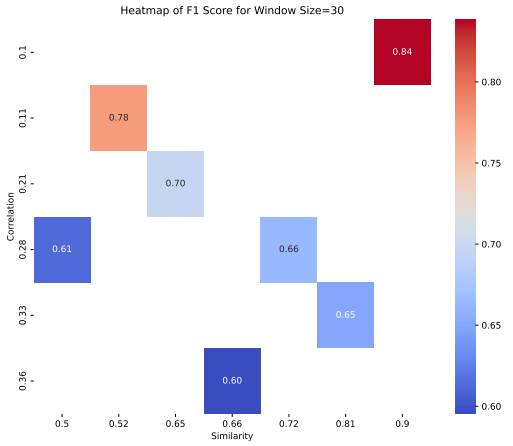

Figure 10: Heatmap of F1 Score with window size = 30 on app recognition

6.3.2 Unseen App Detection

The datasets used in the detection task are partitioned as follows:

- **Training Dataset**: Consists of the training portion of packets from *MAppGraph*.
- **Test Dataset**: Comprises the test part of packets from *MAppGraph* and additional packets from *Cross Platform* sources.

Similar to the app recognition, the performance of the unseen app detection is assessed based on different hyperparameters as well. For the outcomes, Figure 11 shows the

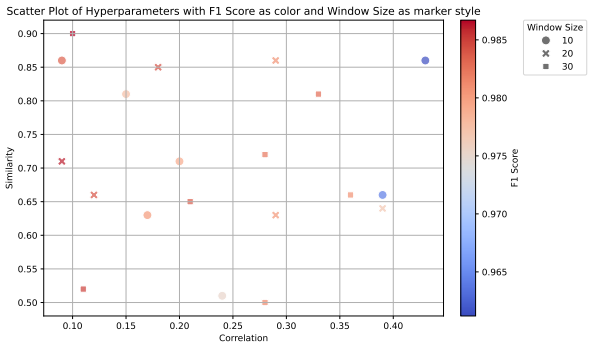

Figure 11: Scatter plot of hyperparameters with F1 Score on unseen app detection

F1 Scores achieved with varying similarity and correlation hyperparameters, using different window sizes as markers. Figure 12 illustrates the F1 Scores in more detailed format.

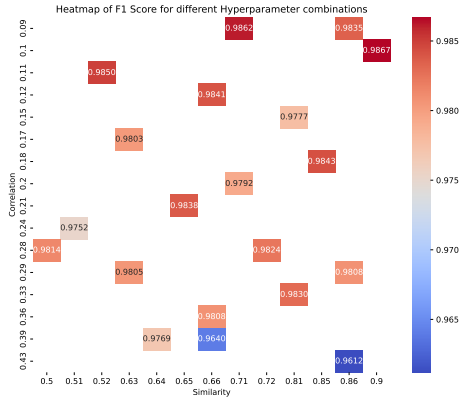

Figure 12: Heatmap of F1 Score on unseen app detection

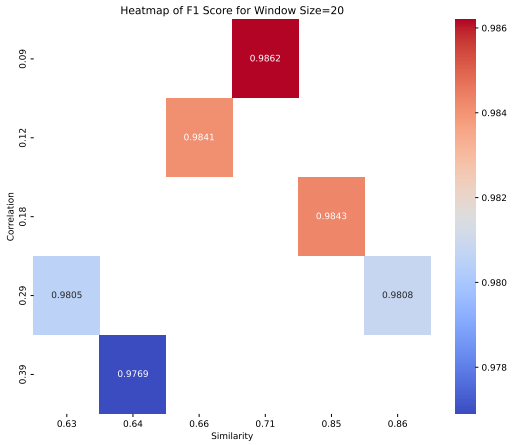

Figure 14: Heatmap of F1 Score on with window size = 20 on app recognition

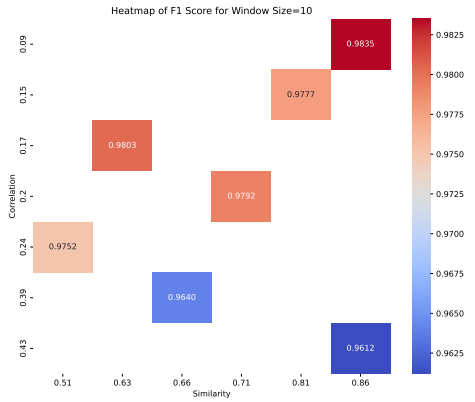

Figure 13: Heatmap of F1 Score on with window size = 10 on app recognition

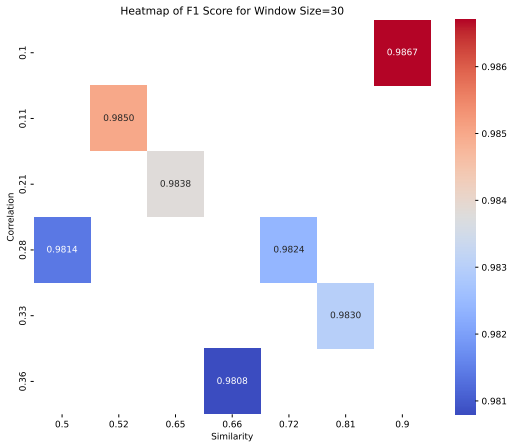

Figure 15: Heatmap of F1 Score with window size = 30 on app recognition

Besides, we use a group of graphs(Figure 13, 14 and 15) to further verify the conclusion that when window size is close to 30, correlation rate is close to 0.1 and similarity rate is closed to 0.9, our new model performs best.

6.4 Model Comparisons

The new model, which excludes DNS flows on port 53, is compared against the original FLOWPRINT model. This comparative analysis showcases the performance gains achieved by the new model, particularly in scenarios with encrypted traffic where DNS queries were previously causing significant noise.

In this section, we still make experiments on both the models' app recognition and detection functions.

6.4.1 App Recognition

- **Training Dataset**: Consists of the training portion of packets from *MAppGraph* and additional packets from *Cross Platform* sources.
- **Test Dataset**: Comprises the test part of packets from *MAppGraph*.

We compares all of the four metrics between the old model and our advanced new model. Based on Figure 16, we could see the new model exhibits improvements in most of the metrics, particularly in precision and F1 score, suggesting enhanced recognition capabilities over the old model.

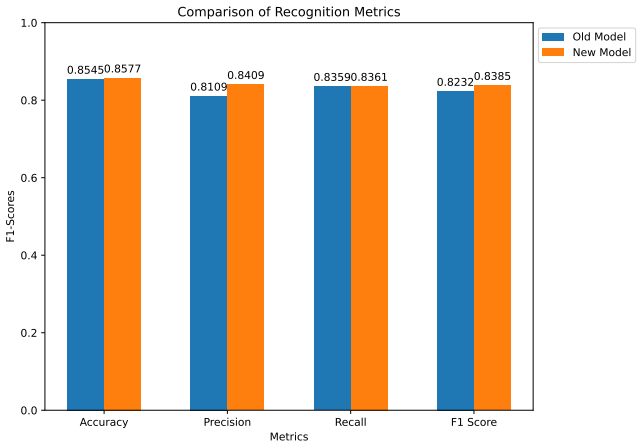

Figure 16: Comparisons on app recognition

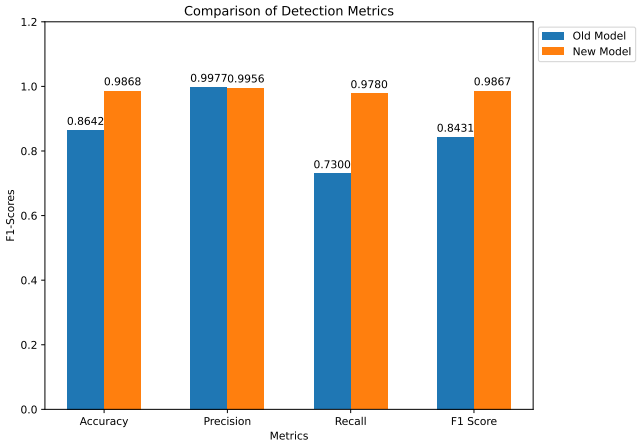

Figure 17: Comparisons on unseen app detection

6.4.2 Unseen App Detection

- **Training Dataset**: Consists of the training portion of packets from *MAppGraph*.
- **Test Dataset**: Comprises the test part of packets from *MAppGraph* and additional packets from *Cross Platform* sources.

Again, we compares all of the four metrics between the old model and our advanced new model. The results in Figure 17 indicate significant improvements in all metrics by the new model, especially in accuracy and precision, reflecting its enhanced capability to detect and classify network activities accurately.

7 Extension on browser

With our work above, we find that by basically using destination tuple of IP and port and TLS certificates, we can identify mobile apps with a high accuracy. Therefore, we would like to apply the model to browser traffic to identify websites running in it.

Here, we do a simple test based on the heuristic envision. We collect packets for 10 websites by burp suite, then we do test on two different situations. The 10 websites includes:

- Music: spotify, zingmp3
- Video: youtube, tiktok, ted, bilibili
- Game: steam, Amongus
- Forum: zhihu
- Image: baiduimage

We collect 10 packets of 10 websites, each for 1-3 minutes, 5000 – 25000 flows. Then we split them into training part and test part, with no intersection.

Then we carry on two different kinds of test. The first is the browser with only one website a time, which is similar to a single app. We expect it to have similar performance as mobile apps. The second is the browser with multiple websites, which is much more complicated and untouched by the model before. We try to evaluate the performances for this complex task as well.

7.1 Browser with a single website

For traffic collected in a browser running a single website, we only do the recognition task to check whether it is similar to tasks on apps. we use train dataset including packets of 95 China apps, and the train part of website packets. Test dataset are test part of the websites package.

The performance shown in 18 is pretty good in this small dataset, with each website are recognized correctly by the extracted fingerprints. The results aligns with our expectation, because single website is similar to one app. For example, youtube.com and youtube app are basically the same.

7.2 Browser with multiple websites

While things become complex with multiple websites. For this part, we use the same train datasets as section 7.1 with two different test sets. We include both recognition and detection tasks in this situation, to check if the model can deal with this kind of data.

The first one is a packet of 3 simultaneous websites,tiktok, youtube, and zingmp3, with 32000 flows for 2.5 minutes. We use the best hyperparameters and get a exact recognition for three websites, as recognition result in 19.

The second one is a packet of 4 simultaneous websites,youtube, amongus, bilibili, and another unseen website in the training dataset, CBS sports. The packet has 53000 flows for 2.5 minutes. We use CBS sports for detection task.

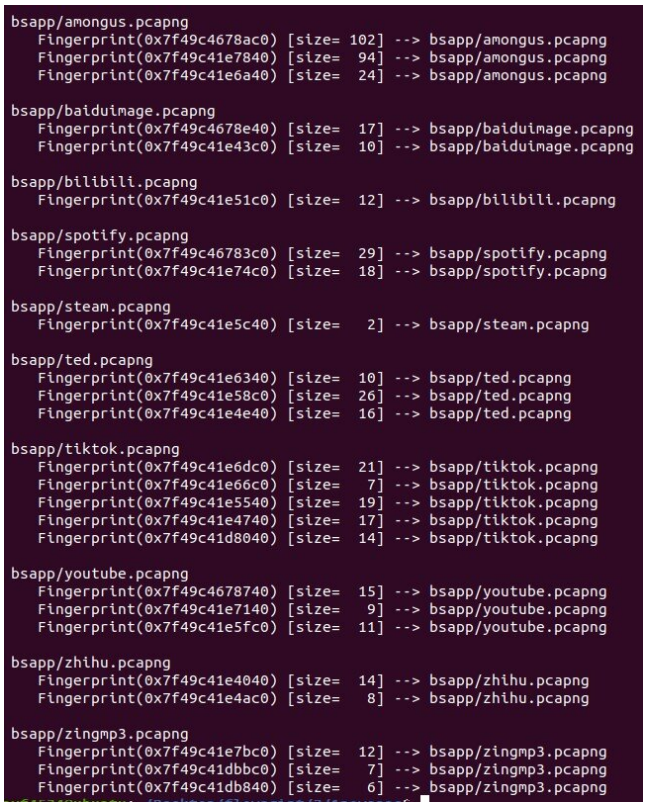

Figure 18: Recognition results of browser traffic with a single website

However, when we use the best hyperparameters, the result is really bad. The fingerprint size is too big, making them mostly detected as unseen. Therefore, we choose another set of hyperparameters with batch size 100, win size 10, detection threshold 0.05 for this, to split fingerprints to smaller one and increase the detection threshold because the large fingerprint size. Then the result in 4 seems to be useful to some extent. It identifies all three known websites, each with one marked red as an example. Meanwhile, it detects unseen website CBS sports at the end, which we interact with at last while collecting data. Therefore, the result is acceptable as the first attempt.

Browser traffic with multiple websites is much more complicated than recognizing a single website or single app, because each fingerprint now are actually mixed by lots of fingerprints from different websites which are active at the same time. While we still find it possible to use this model to give us clues of the websites running in the browser.

8 Challenges

There are many challenges in our work. We solve some of them in our project, while still remains the rest.

First, we have already done amazing task on mobile identification, while it is still difficult to employ the model to a more complex environment. For example, in a browser with multiple websites, its performances are very unstable. We still need more work to make the model work well in these situations.

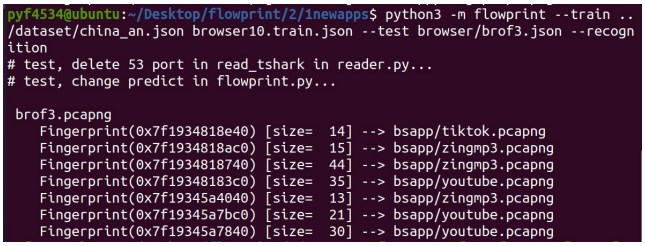

Figure 19: Recognition results of browser traffic with 3 simultaneous website

Next, we still have difficulty in processing data efficiently, especially for more than 100 thousand number of packets. The first step of flow extraction takes more than 90% of total time. If we could makes this process more efficient, it may be possible to be used in some nearly real-time IDS infrastructure.

Meanwhile, although we have gained plenty of packets in this project, it is still a problem to generate sufficient and variant packets data for each app. We also need to consider the influence of different user behaviors during data collection. This is because different people tend to use apps greatly different, which may lead to many different kinds of traffic patterns for the same app. What's more, there are so many different apps in mobile markets, it is a huge work to generate training packets for each app as a result.

9 Conclusion

In this work we improve FLOWPRINT, a proposed approach for creating app fingerprints from the encrypted network traffic of mobile devices. We remove DNS service which may hinder the model's performance. Then the new model achieves a better result in both recognition and detection. We collect apps traffic in 2021 as more updated data in our project. In our evaluation, the new model achieved an accuracy of 85.77% for recognizing apps, a little bit better than 85.54% of the old model. Furthermore, we showed that the new model is able to detect previously unseen apps with a precision of 98.86%, outperforming 86.42% of the old model greatly. These results demonstrate the enhanced capabilities of the new model.

We also extend the usage of the model to browser traffic. It behaves well in recognizing the single website running in a browser. We think the task in this case is quite similar to recognition of one apps, because one website is often a part of corresponding app. While we also experiment on recognition and detection of multiple websites in a browser. This situation

is more complex because multiple fingerprints are now mixed together because of the simultaneous access of all websites. Therefore, there are still a long way to go for this task.

Based on the work now, we have some ideas for the improvements of the model. First, for the model itself, if we degrade weights for common service providers' IP addresses, which can also be learned in some way, such as google or AWS, and possibly increase weights on certain app's TLS certificates, we may have more precise results on recognition and detection. Second, for extension on browser, we can split a mixed browser fingerprints with the help of already learned fingerprints, to detect websites in browser more precisely.

Apps fingerprints are useful in many scenarios. We hope our work and advice can help with research in this field.

Index	Fingerprint	recognition	detection
0	Fingerprint(0x7fb089f68e40) [size= 160]	bsapp/amongus.pcapng	1
1	Fingerprint(0x7fb089f68ac0) [size= 160]	bsapp/amongus.pcapng	1
2	Fingerprint(0x7fb089f68740) [size= 107]	bsapp/amongus.pcapng	1
3	Fingerprint(0x7fb089f683c0) [size= 97]	bsapp/amongus.pcapng	1
4	Fingerprint(0x7fb089cf4040) [size= 153]	bsapp/amongus.pcapng	1
5	Fingerprint(0x7fb089cf7bc0) [size= 89]	bsapp/amongus.pcapng	1
6	Fingerprint(0x7fb089cf7840) [size= 39]	bsapp/youtube.pcapng	1
7	Fingerprint(0x7fb089cf74c0) [size= 67]	bsapp/youtube.pcapng	-1
8	Fingerprint(0x7fb089cf7140) [size= 62]	bsapp/youtube.pcapng	1
9	Fingerprint(0x7fb089cf6dc0) [size= 98]	bsapp/amongus.pcapng	1
10	Fingerprint(0x7fb089cf6a40) [size= 120]	bsapp/amongus.pcapng	-1
11	Fingerprint(0x7fb089cf66c0) [size= 78]	bsapp/amongus.pcapng	1
12	Fingerprint(0x7fb089cf6340) [size= 26]	bsapp/youtube.pcapng	1
13	Fingerprint(0x7fb089cf5fc0) [size= 61]	bsapp/amongus.pcapng	-1
14	Fingerprint(0x7fb089cf5c40) [size= 53]	bsapp/amongus.pcapng	1
15	Fingerprint(0x7fb089cf58c0) [size= 114]	bsapp/amongus.pcapng	1
16	Fingerprint(0x7fb089cf5540) [size= 72]	bsapp/amongus.pcapng	1
17	Fingerprint(0x7fb089cf51c0) [size= 105]	bsapp/amongus.pcapng	1
18	Fingerprint(0x7fb089cf4e40) [size= 102]	bsapp/amongus.pcapng	1
19	Fingerprint(0x7fb089cf4ac0) [size= 110]	bsapp/amongus.pcapng	1
20	Fingerprint(0x7fb089cf4740) [size= 132]	bsapp/amongus.pcapng	-1
21	Fingerprint(0x7fb089cf43c0) [size= 132]	bsapp/amongus.pcapng	1
22	Fingerprint(0x7fb089ce8040) [size= 140]	bsapp/amongus.pcapng	1
23	Fingerprint(0x7fb089cebbc0) [size= 78]	bsapp/amongus.pcapng	-1
24	Fingerprint(0x7fb089ceb840) [size= 132]	bsapp/amongus.pcapng	1
25	Fingerprint(0x7fb089ceb4c0) [size= 38]	bsapp/amongus.pcapng	-1
26	Fingerprint(0x7fb089ceb140) [size= 112]	bsapp/amongus.pcapng	-1
27	Fingerprint(0x7fb089ceadc0) [size= 90]	bsapp/amongus.pcapng	-1
28	Fingerprint(0x7fb089ceaa40) [size= 93]	bsapp/amongus.pcapng	1
29	Fingerprint(0x7fb089cea6c0) [size= 89]	bsapp/amongus.pcapng	1
30	Fingerprint(0x7fb089cea340) [size= 120]	bsapp/amongus.pcapng	1
31	Fingerprint(0x7fb089ce9fc0) [size= 60]	bsapp/amongus.pcapng	1
32	Fingerprint(0x7fb089ce9c40) [size= 33]	bsapp/amongus.pcapng	-1
33	Fingerprint(0x7fb089ce98c0) [size= 11]	bsapp/bilibili.pcapng	1
34	Fingerprint(0x7fb089ce9540) [size= 20]	bsapp/bilibili.pcapng	1
35	Fingerprint(0x7fb089ce91c0) [size= 42]	bsapp/bilibili.pcapng	1
36	Fingerprint(0x7fb089ce8e40) [size= 13]	bsapp/bilibili.pcapng	1
37	Fingerprint(0x7fb089ce8ac0) [size= 25]	pcap/china/android/com.jifen.qukan.pcap	-1
38	Fingerprint(0x7fb089ce8740) [size= 33]	bsapp/steam.pcapng	-1
39	Fingerprint(0x7fb089ce83c0) [size= 97]	bsapp/amongus.pcapng	-1
40	Fingerprint(0x7fb089cdc040) [size= 47]	bsapp/amongus.pcapng	-1
41	Fingerprint(0x7fb089cdfbc0) [size= 37]	bsapp/steam.pcapng	-1

Table 4: Recognition and detection results of browser traffic with 4 simultaneous website

References

[1] 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 *Network and distributed system security symposium (NDSS)*, volume 27, 2020.

[2] 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. IEEE, 2016.

[3] Zeynab Sabahi-Kaviani, Fatemeh Ghassemi, and Zahra Alimadadi. Combining machine and automata learning for network traffic classification. In *International Conference on Topics in Theoretical Computer Science*, pages 17–31. Springer, 2020.

[4] Fatemeh Marzani, Fatemeh Ghassemi, Zeynab Sabahi-Kaviani, Thijs Van Ede, and Maarten Van Steen. Mobile app fingerprinting through automata learning and machine learning. In *2023 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2023.

[5] Thai-Dien Pham, Thien-Lac Ho, Tram Truong-Huu, Tien-Dung Cao, and Hong-Linh Truong. Mappgraph: Mobile-app classification on encrypted network traffic using deep graph convolution neural networks. In *Proceedings of the 37th Annual Computer Security Applications Conference*, pages 1025–1038, 2021.