Trigger Relationship Aware Mobile Traffic Classification

Heyi Tang
tangheyi.09@gmail.com
Tsinghua University

Yong Cui*
cuiyong@tsinghua.edu.cn
Tsinghua University

Jianping Wu
jianping@cernet.edu.cn
Tsinghua University

Xiaowei Yang
xwy@cs.duke.edu
Duke University

Zhenjie Yang
zhenjie.jesse.yang@gmail.com
Tsinghua University

ABSTRACT

Network traffic classification is important to network operators to ensure visibility of traffic. Network management, monitoring, and other services are built upon such classification results for improving quality of service. Compared with traffic classification in non-mobile setting, classification in mobile settings focuses on applications and has become increasingly important. Traditionally, a rule-based method is deployed in a deep packet inspector (DPI) engine for traffic classification. However, with the explosive growth in application usage, the complicated relationships including the use of content delivery networks (CDN) and sharing behaviors among applications make such methods less effective. The traffic may be identified wrongly when one application is connected to another application's server.

In this work, we present **TRAC**: **T**rigger **R**elationship **A**ware traffic **C**lassification, a systematic framework for classifying mobile traffic accurately. In TRAC, we first propose *Trigger Relationship Graph* model to describe the relationships among applications. Then, we introduce a *Trigger Relationship Analyzer* to build the graph based on a modified frequent item set mining method. TRAC classifies the traffic based on the application labels identified by a DPI engine. An *Application Label Corrector* is designed to correct the application labels based on the graph. We evaluate TRAC with one-month data collected from an enterprise network. The evaluation shows that our method can achieve a 17.4% accuracy improvement, from 64.8% to 82.2%.

ACM Reference Format:
Heyi Tang, Yong Cui, Jianping Wu, Xiaowei Yang, and Zhenjie Yang. 2019. Trigger Relationship Aware Mobile Traffic Classification. In *IEEE/ACM International Symposium on Quality of Service (IWQoS '19), June 24–25, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3326285.3329050

*Corresponding Author

IWQoS '19, June 24–25, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6778-3/19/06…$15.00
https://doi.org/10.1145/3326285.3329050

1 INTRODUCTION

Network operators desire to identify an application by the traffic it generates, this problem is called *traffic classification*. Such a problem has become increasingly important due to the growing use of mobile phones, on which a variety of applications are installed. Fine-grained classification of applications is required to achieve improved quality of service (QoS). For instance, a network operator can optimize QoS of certain applications (e.g., cloud gaming [10] and video streaming [40]). Enterprise network operators are also motivated to improve the quality of online meeting. By identifying the applications, firewalls can block known malicious applications or applications incompatible with cooperate policies.

To address this problem, some studies [3, 11, 25, 28, 33] use machine learning approaches to mine statistical features for classification. However, these approaches require much labeled data and are difficult to deploy in practice due to their high computational complexities. Another direction is to find reliable identifiers in network traffic. Numerous studies have attempted to find identifiers in different fields, including HTTP UserAgents [36], Android Manifest.xml [30], URL parameters [37] and domain names [34]. The identifiers are embedded in a rule-based matching algorithm deployed in a DPI engine. Then each TCP flow is associated with an application in the DPI engine. Such methods are more computationally efficient and are widely used in practice.

However, through an in-depth study on a real-world dataset, we find limitations on such methods resulting from the complex relationships among applications. For example, with the rapid growth of cloud services, large numbers of applications are storing their resources in clouds (such as AWS, Azure, and Alibaba Cloud). From the DPI engine's perspective, a flow originating from the cloud is identified as deriving from an application provided by the cloud service. However, the flow is actually triggered by an application using the cloud service. Another typical relationship is sharing among applications: one may share YouTube videos to Facebook, and thus, some of the traffic generated by Facebook will be identified as being from YouTube. The growing use of HTTPS [20] further complicates this problem. This is because the only information that we can obtain from the traffic is the certificate of the source domain (e.g., *.alicdn.com). The techniques that identify traffic by identifiers in payloads do not work any more. In our evaluation with a real-world dataset, a DPI engine with thousands of rules can only correctly identify 64.8% of flows.

In this work, we investigate the above-mentioned problem and find it common in real network traffic. One application may be connected to the server owned by another application, and the

DPI engine may give wrong application label to the flows. We model such relationships among applications as a *Trigger Relationship Graph* to solve the problem. We then present **TRAC**: **T**rigger **R**elationship **A**ware traffic **C**lassification; a framework for classifying the traffic with the information from trigger relationships (§ 2).

In order to build the trigger relationship graph, TRAC runs a *Trigger Relationship Analyzer* offline to find trigger relationships from unlabeled traffic. We propose a relationship mining algorithm to analyze such traffic based on modified frequent item set mining methods (§ 3).

As for classifying the traffic online, TRAC is designed based on an exsiting DPI engine. The DPI engine gives each flow an initial application label, which may be wrong. Then TRAC runs an *Application Label Corrector* to update the initial labels with the trigger relationship graph (§ 4). The updating problem is modeled with maximum a posteriori (MAP) estimate and proved NP-hard. Inspired by the MAP model, we compute application priorities and design a heuristic algorithm for updating the labels and improve accuracy.

Finally, we implement and deploy TRAC framework in an enterprise network (§ 5). Then we collect one month of real-world data to evaluate our system, the results show that TRAC can improve the application classification accuracy from 64.8% to 82.2% (§ 6).

2 TRAC OVERVIEW

In this section, we first discuss the background of traffic classification and the problem caused by the application relationships. Then, we present the trigger relationship model and the design of TRAC framework to solve the problem.

2.1 Traffic Classification Problem

The goal in the traffic classification problem is to give all the TCP flows passing through the router with an application label, so the results can be used for further analysis. The problem is formalized as follows. Given a sequence of TCP flows $\mathcal{F} = \{F_1, ..., F_n\}$, an application set $\mathcal{A} = \{A_1, ..., A_m\}$, for each flow F_i, we assign an application label to F_i as $F_i.label \in \mathcal{A}$.

In practice, the most common method is rule-based classification. In a typical rule-based framework shown in Figure 2a, a *Rule Generator* is run offline to find reliable identifiers and obtain rules. Then, a *DPI Engine* is run online to assign each incoming flow an application label. With the prepared rules generated by the rule generator, We can easily deploy an efficient identifier matching algorithm, such as the Aho–Corasick algorithm [4], in the DPI engine.

2.2 Application Relationships

Although rule-based methods are widely studied [18, 19, 30, 34, 36–38], we find that their accuracy is not satisfactory in practice. Sometimes, a user is using application A, while the DPI engine shows that the user is using both applications A and B. Through the analysis of such applications and traffic, we find that there are two typical scenarios, shown in Figure 1. One scenario is the use of cloud CDN; the other involves the sharing behaviors among application.

(a) App B provides cloud CDN for storing the resources of A and C

(b) The contents from B and C are shared in social network A

Figure 1: These figures show examples of typical scenarios in which the traffic of one application occurs in another application.

The first scenario is that one application fetches data from a cloud server belonging to another application. As Figure 1a shows, Youku is a video service, Weibo is a social network, and Alibaba Cloud provides cloud services for the two applications. When Youku or Weibo is running, some data is fetched from the server with their own domains and can be accurately recognized. However, static resources, such as images and style sheets, are often stored in the CDN, while the CDN servers use the domain of Alibaba. Thus, the traffic is classified as that of Alibaba Cloud wrongly.

The second scenario is that the content of one application is shared in another application. As Figure 1b shows, A user may share videos from YouTube and images from Flicker in Facebook. Then, when the friends of the user browse the news feed of Facebook, contents from both YouTube and Flicker can be viewed in Facebook and generate corresponding traffic. From the perspective of the DPI engine, the traffic includes data from not only Facebook but also YouTube and Flicker. In this scenario, the traffic is also wrongly classified.

Youku, Weibo, YouTube and Flicker are all popular applications, which generate a large amount of traffic in the Internet. These scenarios lead to a serious decline in accuracy.

Besides, there are other scenarios worth discussing. For example, map services are embedded in many other applications, while the traffic is detected as that of a map application. Shopping applications may request third-party payment services. News applications contain HTML links to many other websites. All these scenarios can cause mismatches between traffic classification results and the application being used.

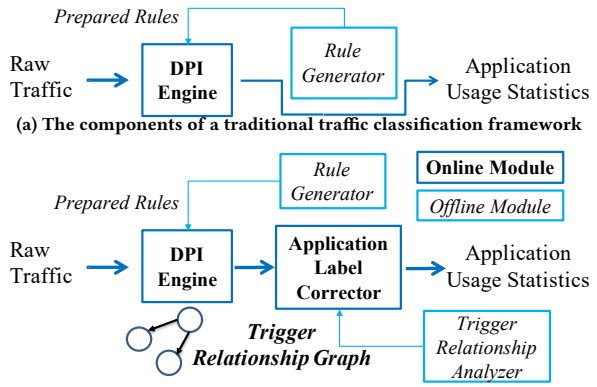
2.3 TRAC Design

According to the observation, we have found the root cause of the decline in accuracy. Here comes a question: *can we avoid such mistakes and improve accuracy?* With the analysis of the scenarios, we define the *Trigger Relationship Graph* and propose the *TRAC* framework to solve the problem.

In the above scenarios, the traffic from wrongly identified applications is in fact triggered by the running application. Thus, we define such a relationship between two applications as a *Trigger Relationship*. All relationships among applications are represented as a *Trigger Relationship Graph*.

Definition 1. Trigger Relationship Graph

Let A and B be two applications. When A is running, if the traffic labeled as B is actually triggered by A, because of the connections between A and the server owned by B, while B is not running, We say that the two applications have a trigger relationship from A to



(a) The components of a traditional traffic classification framework

(b) The components of the TRAC framework

Figure 2: These figures show the traditional traffic classification framework and TRAC framework.

B. We indicate it as $R = (A, B)$ Let $\mathcal{A} = \{A_1, A_2, ...A_m\}$ be the set of all applications. Let $\mathcal{R} = \{(A_{i_1}, A_{j_1}), (A_{i_2}, A_{j_2}), ...(A_{i_3}, A_{j_3})\}$ be the set of all trigger relationships. We define the trigger relationship graph as follows:

$$G = (\mathcal{A}, \mathcal{R})$$

With the trigger relationship graph model, The architecture of TRAC is shown in Figure 2b, two extra components, the *Trigger Relationship Analyzer* and the *Application Label Corrector*, are introduced into the traditional DPI engine architecture. The trigger relationship analyzer attempts to find all trigger relationships among the applications to build a trigger relationship graph. It runs offline and stores the trigger relationship graph. When TRAC classifies the traffic online, the DPI engine first assigns an initial label to each flow, then the application label corrector loads the trigger relationship graph and corrects the application labels. There are some key challenges to address for TRAC to achieve the design goal.

2.4 Challenges of Each Component

2.4.1 Trigger Relationship Analyzer. The key challenges of the trigger relationship analyzer are derived from large number of applications and the lack of information among them.

Specifically, for m applications, each pair of them may have a trigger relationship, and the total number of possible trigger relationships is $O(m^2)$. Taking into account the rapid growth of mobile applications, m would be a very large number and it is constantly increasing. To accurately describe the relationships among these m applications, the search space is undoubtedly enormous.

In addition, we do not have the ground-truth about trigger relationships. The usage of CDN and the sharing between applications are two distinct scenarios of trigger relationships. The simplest way is to analyze such scenarios and list all possibilities with human experts' domain knowledge. However, this cannot really solve the problem, since there may be many other different scenarios where A is connected to B's server. Essentially, we must investigate possible trigger relationships in an unsupervised manner.

2.4.2 Application Label Corrector. The key challenges of the application label corrector include the conversion between flows and applications and the complexity of the trigger relationship graph.

The input of the application label corrector is the sequence of incoming flows, while the existing relationships are among applications. The goal is to update the application labels of the flows; thus, we must establish a bridge between the applications and the flows, thereby converting the relationship between two applications to a relationship between two flows or between the flow and the application.

Another challenge is that the trigger relationship graph is a complicated graph, where all possible applications are in the graph. For one flow, there are many different applications that may have triggered the flow. Thus, we must find the application with the highest probability to trigger such flow.

3 TRIGGER RELATIONSHIP ANALYZER

In this section, we introduce the design of the trigger relationship analyzer. The goal of the trigger relationship analyzer is to build a trigger relationship graph with collected TCP flows. We first describe the workflow of it. Then, the details of each step are described.

3.1 The Workflow

Our core objective in computing the trigger relationship is converting this problem to a frequent item set mining problem [8]. Frequent item set mining is widely used in recommendation systems to find relationships among different item sets. However, our scenario is different from that of the original problem. Thus, our workflow is aimed to convert our problem to a similar form to frequent item set mining.

The workflow of the trigger relationship analyzer is shown in Figure 3. First, we convert TCP flows to multiple application sets so that the applications take on a form similar to the frequent item set mining problem. Thus the relationships among applications can be mined from the sets.

Given numerous application sets, we propose a trigger relationship mining algorithm consisting of the following three methods. At first, we define association relationships mined from the application sets under some constraints. Association relationships under strict constraints are treated as trigger relationships. Second, with an application executor, we obtain application sets with known labels and compute trigger relationships from such application sets. The last method is determining uncertain relationships using domain knowledges obtained from human experts.

By doing so, we obtain enough trigger relationships among the applications. The relationships thus constitute a full trigger relationship graph.

3.2 Converting TCP Flows to Application Sets

First, we need to convert the TCP flows to application sets. Considering that if A triggers B, both the traffic labeled as A and B can be observed at the same time. Thus we cluster the TCP flows into multiple sets using proper time intervals. We use an existing DPI engine to assign an application label $F_i.label_{DPI} \in \mathcal{A}$ for each flow. The application labels of those flows can consist of an application set in each period. An application set of a specific period is as follows:

$$S_{t,t+\Delta t} = \{F_i.label_{DPI}, F_i.t_{start} < t + \Delta t, F_i.t_{end} > t\}$$

With a proper time interval, we can easily traverse all the TCP flows and generate many application sets.

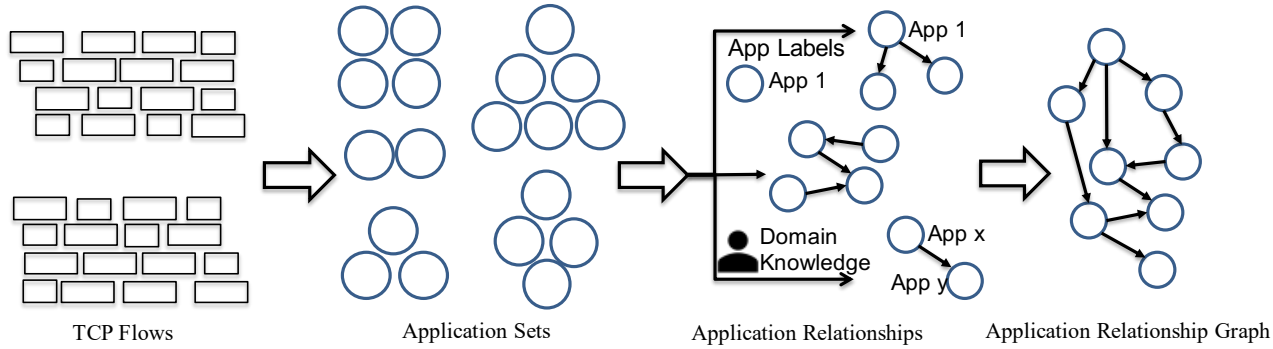


Figure 3: This figure shows the steps of the trigger relationship analyzer

3.3 Relationships without Information

In the original frequent item set problem, the object is to compute relationships among sets of items with *support* and *confidence*. Our problem aims at analyzing the relationships between every two applications. Inspired by the definitions in frequent item set problem, we propose the following definitions in our context.

Definition 2. Support

The support of an application A is defined as the number of the application sets in the application set database $\mathcal{S} = \{S_1, S_2, ...S_n\}$ that contain A. Therefore, the support $sup(A)$ is defined as:

$$sup(A) = |\{S, S \in \mathcal{S}, A \in S\}|$$

The support of two applications A, B requires S to contain both A and B. Then it is defined as:

$$sup(A, B) = |\{S, S \in \mathcal{S}, A \in S, B \in S\}|$$

Definition 3. Confidence

Let A and B be two applications. The confidence $conf(A \Rightarrow B)$ of the relationship $A \Rightarrow B$ is the conditional probability of A and B occurring in all applications sets given that the application sets contain A. Therefore, the confidence $conf(A \Rightarrow B)$ is defined as:

$$conf(A \Rightarrow B) = \frac{sup(A, B)}{sup(A)}$$

Definition 4. Association Relationship

Let A and B be two applications. The relationship $A \Rightarrow B$ is said to be an association relationship with a minimum support of $minsup$ and minimum confidence of $minconf$ if it satisfies both of the following criteria:

$$sup(A, B) \geq minsup, conf(A \Rightarrow B) \geq minconf$$

When any other information is absent, we compute the association relationship between every pair of applications which have occurred in the same application set. The association relationships can be treated as trigger relationships if we use large $minsup$ and $minconf$.

3.4 Relationships from Genarated Flows

However, we have observed that association relationships do not always equal trigger relationships. For example, assuming that a user does not install a video application A, while she usually uses social network application B. The videos in A can be shared in B, so the user can sometimes watch videos from A in B. As a result, whenever A occurs in an application set, B will also occur since all As are embedded in B. With the above definitions, we will obtain the association relationship $A \Rightarrow B$. However, the direction of the trigger relationship is actually from B to A. Thus, we need additional techniques to compute more accurate relationships. We then use generated flows for determining the correct direction.

First, we use an application executor to automatically execute the known applications and capture the generated flows. The executor runs each application and simulates user behaviors to generate network traffic similar to real traffic. Android automation tools, such as monkey-runner [1], are used to execute Android executables. Our program in such executables checks user interface (UI) elements in the application and generate UI events for simulating real users. The traffic is captured on a router connected to the device.

The flows are then processed with the above-mentioned techniques and converted to multiple application sets. Thus, we have application sets with corresponding correct application labels. A simple method is to let every application occurring in the set have a trigger relationship with the correct application.

However, there are other applications running in the background. This approach is not satisfactory since there may be no relationship between background applications and the correct application. Considering that the background applications are not always running, thus we take the *support* of the occurring applications into consideration and use item set mining in the labeled scenario.

For each application A, we constrain the mining space to the application sets with label A. Thus, we have an application set database $\mathcal{S}_A$. We then define the support of application B on A as $sup_A(B)$. The confidence of B on A is defined as:

$$conf_A(B) = conf_A(B \Rightarrow A) = \frac{sup_A(A, B)}{sup_A(B)}$$

With such definition, we then define the trigger set of A.

Definition 5. Trigger Set of A

The trigger set of A is defined as the set of applications which could be triggered by A. By computing the support and the confidence of applications on $\mathcal{S}_A$, the trigger set is formalized as follows.

$$T_A = \{B, sup_A(B) \geq minsup, conf_A(B) \geq minconf\}$$

Only the applications which are highly relevant to A are added to the T_A. The background applications are ignored.

3.5 Relationships from Domain Knowledge

However, due to the lack of data, the above-mentioned techniques cannot find all trigger relationships. The requirement for an association relationship is strict; the constraints on both $minsup$ and $minconf$ should be satisfied. Certain trigger relationships may only satisfy one constraint.

To find as many trigger relationships as possible, we need to check the details of the relationships among applications. We list

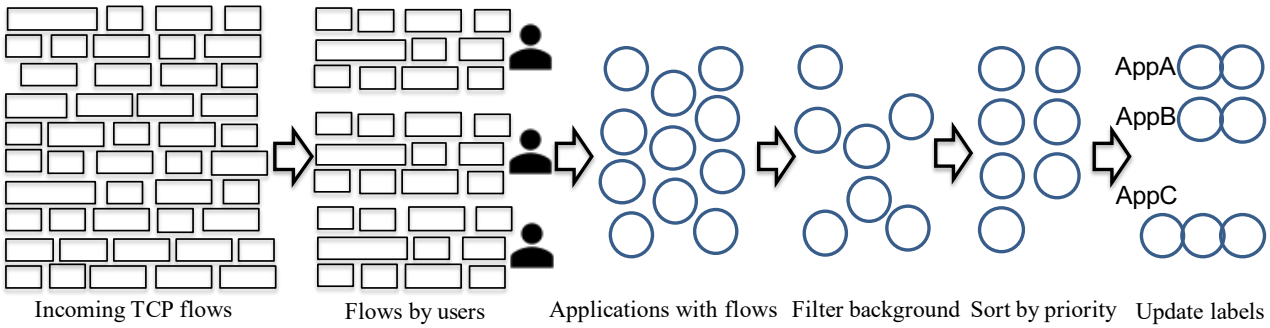


Figure 4: This figure shows the steps of the application label corrector

relationships that only satisfy one constraint. Then, we let human experts examine such relationships and label them as true or false with domain knowledges.

For example, WeChat is the most popular application in China, and Tencent Cloud is a large cloud provider in China. Both traffic labeled as WeChat and traffic labeled as Tencent Cloud occur many times in our dataset. We denote WeChat with A_w and Tencent Cloud with A_{tc}. The result is that $sup(A_w, A_{tc})$ has a large value, while both $conf(A_w \Rightarrow A_{tc})$ and $conf(A_{tc} \Rightarrow A_w)$ are no greater than $minconf$ since $sup(A_w)$ and $sup(A_{tc})$ are too large. Thus, the frequent item set mining algorithm cannot determine whether these two applications have a trigger relationship. However, given domain knowledge, we know that WeChat is also an application belonging to Tencent Inc, while all the applications of Tencent Inc will use Tencent Cloud as their CDN service. Thus, we can conclude that a trigger relationship exists from WeChat to Tencent Cloud.

3.6 Trigger Relationship Mining Algorithm

The methods for mining the trigger relationship are summarized as algorithm 1. At first, the trigger relationships mined from $\mathcal{S}_{labeled}$ are added to the graph in line 4. Then all possible relationships mined from $\mathcal{S}$ are enumerated, if such relationship has been added in the first step, then the relationship is passed. Otherwise, the relationships that satisfy both constraints are added, the relationships that satisfy none of the constraint are dropped, the remaining relationships are sent to human experts for further checking.

Algorithm 1 Trigger Relationship Mining

Require: Unlabeled Application Set Database $\mathcal{S}$
Require: Labeled Databases $\mathcal{S}_{labeled} = \mathcal{S}_{A_1}, \mathcal{S}_{A_2}, ...\mathcal{S}_{A_k}$
Ensure: Trigger Relationship Graph $G = (\mathcal{A}, \mathcal{R})$
1: **for** $\mathcal{S}_{A_i}$ in $\mathcal{S}_{labeled}$ **do**
2: Compute the trigger set T_{A_i} with $\mathcal{S}_{A_i}$
3: **for** A in T_{A_i} **do**
4: Add (A_i, A) to $\mathcal{R}$
5: **end for**
6: **end for**
7: **for** (A_i, A_j) IN all possible relationships from $\mathcal{S}$ **do**
8: **if** NOT $((A_i, A_j)$ IN $\mathcal{R}$ OR (A_j, A_i) IN $\mathcal{R})$ **then**
9: **if** $sup(A_i, A_j) \geq minsup$ AND $conf(A_i \Rightarrow A_j) \geq minconf$ **then**
10: Add (A_i, A_j) to $\mathcal{R}$
11: **else if** $sup(A_i, A_j) \geq minsup$ OR $conf(A_i \Rightarrow A_j) \geq minconf$ **then**
12: Store (A_i, A_j) for checking by human experts
13: **end if**
14: **end if**
15: **end for**

4 APPLICATION LABEL CORRECTOR

With the trigger relationship analyzer, we compute a trigger relationship graph with multiple trigger relationships among applications. The next question we aim to answer is: how can we use the trigger relationship graph to correct the initial labels in the application label corrector? We first convert incoming TCP flows to a running application set. Then we predict which application should be updated to another application and propose an application label updating algorithm. The detailed workflow and the steps are described in this section.

4.1 The Workflow

The application label corrector is run after the DPI engine and aim to assign a correct new application label to each TCP flow. The problem is formulated as follows: We are given a sequence of TCP flows $\mathcal{F} = \{F_1, F_2, ...F_n\}$, in which the following information can be used:

- The initial application label of the flow $F_i.label_{DPI}$, which is labeled by the existing DPI engine;
- The start timestamp and end timestamp of the flow;
- The total traffic size of the flow;
- Other detailed information contained in TCP headers;

Our goal is to compute the correct application label $F_i.label$ for each flow, such that the accuracy of the architecture can be improved. The workflow of the application label corrector is shown in Figure 4. There are five key steps in updating the application labels of the flows.

Since the TCP flows are collected in the router, those flows include the flows generated by different users. The relationships between two applications are only meaningful in the same user's device. Therefore, we first need to cluster the flows into different sets for different users.

For the flows of each user, as an online system, we collect flows from the previous n seconds as "the traffic generated by the currently running application". Then, the flows from each application are combined into one flow indicating such application, with the number of flows and the total size.

There may be applications running in the background. Such applications may affect the updating algorithm that we will mention later. Thus, we should then filter the background applications before updating the flows to guarantee accuracy.

The next step is to update some applications as other applications according to the trigger relationship graph, which means that such applications are triggered by the correct applications. This problem is formalized as a MAP (maximum a posteriori) estimate problem [26]. With our analysis, the information we observed is that, if A triggers B, and A, B are both observed in the traffic, only A is actually running, the traffic of B is triggered by A.

According to this observation, we define application priorities according to the trigger relationship graph. Then we design a priority based application label updating algorithm to obtain a mapping between applications, and finally use it to update all the flows.

4.2 Clustering Flows by User

The TCP flows are captured by the router and are labeled by the DPI engine. To update the labels according to the trigger relationships, we first need to cluster the TCP flows into different sets by user identifier. The reason is that if an application is triggered by another application, the two applications must be on one device of the same user. Two flows from different devices will not be related.

To cluster the TCP flows, we use IP addresses as user identifiers. In practice, a traffic classification device is usually deployed in an enterprise network. The network operator knows the range of IP addresses in the enterprise network. Thus, it is easy to maintain an IP set for the enterprise network. If either the source address or the destination address of a flow matches an IP in the IP set, the flow is assigned to the corresponding IP's group.

There are two minor issues that we need to consider. If the enterprise network uses DHCP for allocating IP addresses, the IP of each user may change over time. We do not believe that this approach affects the result. Because the IPs do not change very frequently. Over a short period, the flows from the same IP belong to the same device; thus, our system will be effective.

The other issue is that the enterprise network may use NAT. Two users may share the same IP address. To distinguish different users using the same IP addresses, the port mapping in the NAT devices should be taken into consideration when assigning flows to users.

4.3 The Flow Set in Previous Seconds

After assigning the flows, we address each flow on a user-by-user basis. For each user, we should collect the flows in the previous Δt seconds. We assume that over a short period, a user will not use many different applications. Therefore, the flows generated over a certain period are considered simultaneously generated, and some are triggered by others.

To satisfy the performance requirements as an online system, we use a sliding window approach to collect the flows in the same time period. The sliding window records the initial time stamp once a flow occurs. After a certain period has passed, the sliding window sends all the flows that are currently recorded to the next step and removes all flows that have ended. Then, the sliding window starts recoding the flows in the next period. The collected flows in the last period are then processed in subsequent steps.

4.4 Background Application Filter

With the original flows, the background application filter now has a set of applications with its flow count and the total traffic. We have observed that background traffic exists in the flows in practice. An application may run in the background and send heartbeat messages, but the application is in fact not running, although the traffic is being captured. Such traffic may affect the accuracy since it may also exist in the trigger relationship graph. According to algorithm 2 we will mention later, an application A which may trigger other applications, may make some flows update their application labels to A, even though A is not actually running in the foreground.

Since most traffic from background applications is heartbeat traffic, the number of flows and the total traffic are not excessive. A simple way to filter out the background applications is ignoring the applications whose flow count and traffic size do not reach a certain threshold. The threshold is learned with the labeled traffic.

With the background application filter, we finally have a set containing the applications we observe in foreground traffic.

4.5 MAP Model and Application Priority

With such an application set that we observe in the traffic, our goal is to predict that, which applications are really running, and which applications are triggered by other applications. Here we formalize the problem as an MAP estimate problem as follows.

Definition 6. Application Predicting (AP) Problem

Given an application set $A_1, A_2, ...A_m$, let $x_i \in \{0, 1\}$ to indicate whether A_i is running, let $y_i \in \{0, 1\}$ to indicate whether we observe A_i in the traffic. The vector $y = (y_1, y_2, ...y_m)$ indicates the application set in the traffic, $x = (x_1, x_2, ...x_m)$ indicates the set of applications which are actually running. Let $P(x_i = 1) = p_i$, A_i and A_j are independent. $P(y_i = 1|x_i = 1) = 1$, a running application must be observed. A trigger relationship is denoted by $P(y_j = 1|y_i = 1) = p_{ij}$. The AP problem is to find a x' to maximize the posteriori probability $P(x = x'|y = y')$, for a given $y = y'$, and the above prior distribution of x.

Lemma 1. *AP Problem is NP-hard.*

Proof. We prove this lemma by reducing integer linear programming [24] to the AP problem. Consider a simplified case that A_1 triggers $A_i, \forall i > 1$, while any other A_i and A_j are independent. In this case we let $y'_i = 1, \forall i \leq m$, then $P(x = x'|y = y') = \Sigma_{i=2}^{m}(x'_i p_i + (1 - x'_i)(1 - p_i)p_{1i})$. Maximizing the value is a integer linear programming problem. Thus AP problem is NP-hard. □

We then use a simplified case to inspire us for a heuristic algorithm. The expected number of running application is only one, thus we assume $p_i = 1/m, \forall i$. A typical case is that, A_i and A_j are observed, while there is a trigger relationship from A_i to A_j. We use Y_{11} to denote the event $y_i = 1, y_j = 1$, similarly we use X_{10} to denote the event $x_i = 1, x_j = 0$. We have the following equations.

$$P(Y_{11}) = P(Y_{11}|X_{10}) \cdot P(X_{10}) + P(Y_{11}|X_{11}) \cdot P(X_{11})$$

$$= p_{ij} \cdot (m-1)/m^2 + 1 \cdot 1/m^2 = (p_{ij}(m-1) + 1)/m^2$$

$$P(X_{10}|Y_{11}) = \frac{P(Y_{11}|X_{10}) \cdot P(X_{10})}{P(Y_{11})} = \frac{p_{ij}(m-1)}{p_{ij}(m-1) + 1}$$

$$P(X_{11}|Y_{11}) = \frac{P(Y_{11}|X_{11}) \cdot P(X_{11})}{P(Y_{11})} = \frac{1}{p_{ij}(m-1) + 1}$$

$$P(X_{10}|Y_{11}) > P(X_{11}|Y_{11}) \Leftrightarrow p_{ij} > 1/(m-1)$$

The equations show that, if p_{ij} is greater than $1/(m-1)$, only A_i is actually running, thus A_j is triggered by A_i, the flows labeled as A_j should be updated to A_i. In practice, m is a large value, since there are a lot of applications a user may install, $p_{ij} > 1/(m-1)$ is always correct. The application that could be triggered by other applications should be updated.

With the above-mentioned observations, we are able to update some applications to others. Then we need to determine the order of the applications to update them. Each application should be set priority values for comparison to other applications. If the priority of application A is greater than the priority of application B, A has a higher probability to be the running application and a lower probability to be an application triggered by other applications. Based on this definition, clearly, when there is a trigger relationship from A to B, A has a higher priority.

This observation suggests that, we can use topological sorting [12] to compute the priorities of the applications. Topological sorting on a directed acyclic graph (DAG) is a linear ordering of vertices such that, for every directed edge uv, vertex u comes before v in the ordering. In our scenario, the ordering of applications after topological sorting directly represents the priorities of the applications.

Since the trigger relationship graph is computed offline in the trigger relationship analyzer, the priority of the applications can also be computed offline. In the online phase, we directly use these priorities to sort the applications.

Another issue is that topological sorting is only valid in a DAG, whereas the trigger relationship graph is simply a directed graph. In practice, when generating the relationships in the trigger relationship analyzer, if the relationships A to B and B to A both exist, we remove one of the relationships according to domain knowledge. After addressing all such edges, the graph no longer contains cycles in our data.

4.6 Updating Application Labels

With the priorities of the applications, we determine the order of the applications in the current application set. Then, we should compute which applications are triggered by other applications so that we can update the labels of the corresponding flows.

One concern is that there may be triggering chains: application A triggers B, while B triggers C. For example, the content in application B is shared in application A, while application B is using the CDN service of application C. The relationships (A, B) and (B, C) are both in the trigger relationship graph, but (A, C) is not in the graph since A and C do not have a direct relationship. However, concerning the traffic, the flows from both B and C are triggered by A, and they both need to be updated. In our algorithm, we use a hash map H_t to address it, where $H_t[A_i]$ stores the applications in the current set that is triggered by A_i.

The details of our algorithm are shown in Algorithm 2. We use H_{label} to store the new application label indicating that an application should be updated. In line 4, we sort the applications in increasing order of priority. Thus, the applications with lower priority are computed first. For each application A_i, we find application A_j that can trigger it with the highest priority in line 7. Once such A_j is found, $H_{label}[A_i]$ is updated, and A_i is added to $H_t[A_j]$ to address the triggering chain case. In lines 11 and 12, we update all applications in $H_t[A_i]$. Considering that this procedure is run in an iterative manner, all applications in the triggering chain are updated to the root application in the chain. If A_i has not been updated with any A_j, which means that A_i is the running application, not triggered by any other application, we set $H_{label}[A_i]$ to itself in line 18.

Algorithm 2 Application Label Updating

Require: Observed Application Set $A_1, A_2, ...A_n$
Require: Trigger Relationship Graph $G = (\mathcal{A}, \mathcal{R})$
Ensure: An hash map H_{label}, storing the new label.
1: Let H_{label} be an empty hash map.
2: Let H_t be empty to store the apps triggered by A_i.
3: Construct application list $L = [A_1, A_2, ...A_n]$.
4: Sort L with the priorities in increasing order.
5: **for** i = 1 TO n **do**
6: **for** $j = n$ TO $i + 1$ **do**
7: **if** $(L[j], L[i])$ IN $G.\mathcal{R}$ **then**
8: $H_{label}[L[i]] = L[j]$
9: $H_t[L[j]].add(L[i])$
10: **for** A IN $H_t[L[i]]$ **do**
11: $H_{label}[A] = L[j]$
12: $H_t[L[j]].add(A)$
13: **end for**
14: BREAK
15: **end if**
16: **end for**
17: **if** NOT $L[i]$ IN H_{label} **then**
18: $H_{label}[L[i]] = L[i]$
19: **end if**
20: **end for**

After running the application label updating algorithm, for each flow in the flow set, we can simply set

$$F_i.label = H_{label}[F_i.label_{DPI}]$$

Thus, the incoming flows are updated to new applications.

The complexity of the algorithm is $O(n^2)$. Although there are 3 loops, the loop in lines 11 and 12 only runs once in each iteration. Since the count of observed application set n is not large, usually less than 10 in practice, the algorithm represents a small portion of the total running time in TRAC framework.

5 IMPLEMENTATION AND DEPLOYMENT

TRAC is implemented and deployed in a real-world enterprise network for application usage statistics. In this section, we describe our implementation for online classification in the context of the real-world scenario. The architecture is shown in full in Figure. 5. The offline modules including the trigger relationship analyzer and the rule generator are run offline and not shown in the figure.

5.1 Traffic Aggregation

The DPI engine is deployed on the gateway router so that the traffic of all users can be processed. Our first problem is that, the router should send the classification results to the DPI engine, but building one connection for each packet is too costly. We use traffic aggregation to solve this problem. The packets are recorded continuously until they amount to 1 GB of data and then sent to the DPI engine. The output of the DPI engine includes multiple records, where

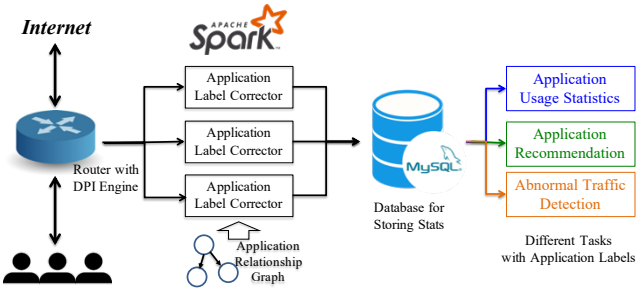


Figure 5: This figure shows the deployment in practice

each record contains different types of information, as described in Section 4. The records are then stored in HDFS [9] for further processing.

5.2 Parallelized Application Label Corrector

The application label corrector read the information of flows from HDFS for updating the labels. Here comes the second problem: although processing one user's is efficient, there are a lot of users' traffic passing through the router. To accelerate the updating process, we use Spark [39] to process the flows in parallel. With the hive keyword "GROUP BY", the flows are divided into several groups by their user identifiers, and different groups of data are sent to corresponding spark nodes.

In practice, the information of the flows is presented as text files, they are stored in a specific folder named "unfinished" in each Spark node. The corrector on each node scans the folder and loads the flows in the files. Then, the corrector assigns flows new application labels and stores them in a MySQL cluster. The original text files in the "unfinished" folder are then removed.

5.3 Storage and Usage Analysis

We use a MySQL [2] cluster to store the statistics of the applications. Here we face the final problem: storing the detailed information of all flows costs a lot of storage resources. Too many records in the database also reduce its performance. Thus, the corrector combines multiple flows into one record for each hour, each record only includes the total traffic and total running time of an application in one hour. The database stores such record, the storage cost is substantially reduced. In this manner, researchers seeking to conduct further studies, such as a statistical analysis of application usage, can fetch the results from the database efficiently.

6 EVALUATION

6.1 Dataset

We use one-month-long real-world data for evaluation. The data was collected in January, 2019 in an enterprise network. A total of three individuals volunteered. Their mobile phones were connected to the router in the enterprise network. Each day, they recorded the applications that they actually used, and the records were collected with their captured traffic. The records are used as the ground-truth for evaluation.

Each flow is labeled with an application label in TRAC. If the application label is in the corresponding records, we say that the flow is correctly identified; otherwise, the flow is not correctly identified.

6.2 Accuracy in Trigger Relationships

We first evaluate the performance of the trigger relationship analyzer. We choose 100 most popular applications in China and run an application executor for generating $\mathcal{S}_{labeled}$ mentioned in S 3.6. The traffic in the one-month-long data is used for generating $\mathcal{S}$. All mined trigger relationships are manually checked by the human experts with domain knowledge. As mentioned in S 2.2, there exist other different trigger relationships that we don't know, thus a relationship may be true, false or uncertain.

$minsup$	$minconf$	Count	True	False	Uncertain
10	0.5	20	85.0%	10.0%	5.0%
10	0.1	104	51.9%	17.3%	30.8%
2	0.5	68	39.7%	5.9%	54.4%

Table 1: This table shows how the parameters affect the count of relationships the trigger relationship analyzer finds.

Totally 80 trigger relationships are mined from the databases from 41 applications in $\mathcal{S}_{labeled}$. 68 of them are true while only 5 of them are false. No relationship is mined from the databases from other 59 applications. As for the the relationships from $\mathcal{S}$, the results with different parameters are shown in table 1. When we let $minsup = 10$ and $minconf = 0.5$, only 20 trigger relationships are found, but they reach a high accuracy that 17 of them are true while only 2 of them are false. When we use lower $minsup$ or lower $minconf$, more relationships can be found, while the accuracy decreases. An interesting fact is that, only about 10% of the relationships are overlapped in two methods, they can complement each other.

As a result, the unsupervised methods, including mining from $\mathcal{S}_{labeled}$ and mining from $\mathcal{S}$ with strict constraints, can achieve high accuracy. The relationships from $\mathcal{S}$ with loose constraints can also be used with domain knowledges. Finally the analyzer builds a trigger relationship graph with 162 edges and stores it for online classification. Note that more than 30% of relationships found in the methods are uncertain. Therefore, more scenarios of trigger relationships should be investigated.

6.3 Classification Accuracy

With such a trigger relationship graph, we then evaluate the classification accuracy of TRAC. We compare TRAC to the traditional rule-based DPI engine described in section 2. As for TRAC, except for the framework with a complete application label corrector, we also evaluate TRAC without filtering background applications in the application label corrector to gain more knowledge of the framework. Since the size of each flow in different applications are different, we evaluate the accuracy in both flow count and traffic size to make the evaluation more convincing.

The one-month-long data includes the data in weekdays of four weeks. The behaviors of users may vary from week to week, so we also evaluate the methods in four weeks respectively. Figure 6 shows the statistics of the data. There are 38090 flows in total, the size of the traffic is about 9.3 GB in total. The users have the most activities in the second week, about 5.7 GB traffic are captured in this week, which accounts for about 2/3 of the total. However, the flow count in the second week is 15889, only accounts for about 2/5 of the total. This is because one volunteer has viewed more videos in the second week.

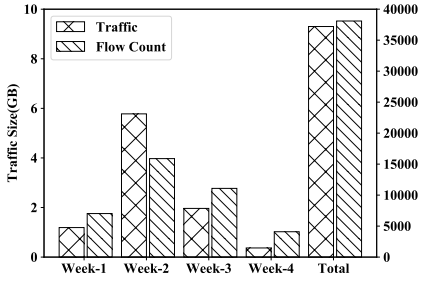


Figure 6: This figure shows the traffic size and the flow count in our dataset

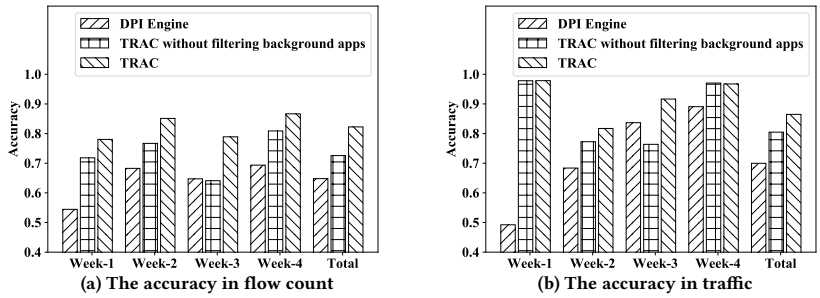


Figure 7: These figures show the accuracy of our method compared to rule-based DPI engine in our one-month dataset.

The accuracy also varies from week to week. As shown in Figure 7a and Figure 7b, The accuracy in both flow count and traffic size has a significant improvement. The accuracy of the DPI engine is 64.8% in flow count and TRAC reaches 82.2%. As for traffic size, the accuracy is improved from 70.0% to 86.5%. The application label corrector successfully finds out plenty of flows which are triggered by other applications. In section 4.4, we have discussed the effects of the background application traffic. The results show that, without the background application filter, the accuracy in flow count can only reach 72.6%. In the third week, the accuracy even drops down when we don't use the filter. Some of the flows correctly labeled in the DPI engine are updated to an application that is running in the background.

Another interesting fact is that, the accuracy in flow count is more stable than the accuracy in traffic size. All the values of the accuracy in flow count in the four weeks are between 78%-86%, while the values of the accuracy in traffic size are from 81%-97%. This difference is caused by the imbalance among different applications in traffic size. Some applications have fewer network activities, of which the flows only contain some bytes of texts, while the social network applications may fetch a lot of images. In the first week and the fourth week, the applications which generates most traffic are labeled correctly, so the accuracy in traffic size reach more than 95%, while the accuracy in flow count is still about 80%.

6.4 Typical Trigger Relationships

Table 2 shows some of the trigger relationships found by TRAC. Their percentages of all flows are also shown in this table. There are 46 trigger relationships seen in the one-month-long data, the table presents several typical relationships.

The table shows that the most common trigger relationship is the use of cloud CDN. Alibaba cloud is the most common cloud CDN in the dataset. Several different applications owned by different companies are using the same CDN from Alibaba Cloud. This result is consistent with the fact that Alibaba Cloud is the largest cloud service provider in China. Tencent Cloud is usually used by the applications owned by Tencent Inc. Because of the widely use of Wechat, the most popular IM application in China, the trigger relationship from Wechat to Tencent Cloud is the most common in the data.

Except for the relationship of cloud CDN, other kinds of relationships are also seen in the data. The volunteers view videos shared from Miaopai in Sina Weibo. Wechat provides a service that we can check QQ Mail in it, this case is also seen in the data.

Triggered App	Running App	Percentage	Relationship Type
Alibaba Cloud	Sina Weibo	1.29%	CDN
Alibaba Cloud	Youku	0.93%	CDN
Alibaba Cloud	Xiami Music	0.90%	CDN
Alibaba Cloud	Auto Navi	0.72%	CDN
Tencent Cloud	Wechat	4.79%	CDN
Tencent Cloud	Tencent Video	0.55%	CDN
Miaopai	Sina Weibo	0.40%	Sharing
QQ Mail	Wechat	0.44%	Service
Baidu Map	Ctrip	0.04%	Service

Table 2: This table shows some typical trigger relationships found by TRAC in our evaluation.

7 RELATED WORK

Traffic classification has been investigated for network management tasks. Several different techniques on traditional traffic classification are summarized in [13]. With the growing usage of mobile devices, recent studies focus on identifying the mobile traffic in application granularity. A survey of these works can be found in [29]. Most researches [18, 19, 34, 36–38] find reliable identifiers for building rule-based algorithms at a per-flow granularity. For instance, the identifiers in HTTP User-Agents are investigated in [36]. AppPrint [18] and FLOWR [37] both extract signatures from the URL in HTTP traffic. SAMPLES [38] proposes a systematic framework that finds the identifiers in different fields in an automated fashion. AMPLES [19] further enhances the identifier matching with fuzzy match. In practice, the identifiers in these studies can be used to enhance a DPI engine for identifying applications. However, such methods do not consider relationships among applications, thus, the accuracy will reach a limit since more and more applications use cloud CDNs.

Recently, numerous studies [3, 5–7, 11, 14, 22, 25, 27, 28, 31–33] have attempted to use statistical features of the flows and leverage machine learning methods. AppScanner [28] focuses on encrypted traffic and tries different supervised machine learning methods including SVM and random forest. Aceto1 et al. [3] try several different deep learning models and use first N bytes of a packet as its feature. These studies show good results, but they are costly thus hard to deploy in practice.

With the results of mobile traffic classification, researchers can conduct further studies [15–17, 21, 23, 35]. MAPPER [23] proposes a system that provides system administrators capabilities of enforcing policies on mobile traffic. Kuo et al. [17] investigate the relationship between application usages and the design of backend systems. Jiang et al. [15] try predicting the future application usage for improving user's experience. Thus traffic classification frameworks with high accuracy are required and our work just meets the actual need.

8 CONCLUSION

In this paper, we investigate the traffic classification problem in practice. We find out that the applications have complicated relationships with each other, which greatly affect the accuracy of traffic classification. We design the TRAC framework with a trigger relationship model to solve the problem and improve accuracy. The evaluation on a real world data set shows that TRAC achieves a significant improvement in accuracy.

ACKNOWLEDGMENTS

This work is supported by NSFC (No. 61872211) and National Key R&D Program of China under Grant 2017YFB1010002.

REFERENCES

[1] [n.d.]. Monkeyrunner. developer.android.com/studio/test/monkeyrunner/.
[2] [n.d.]. MySQL. https://www.mysql.com/.
[3] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé. 2018. Mobile Encrypted Traffic Classification Using Deep Learning. In *TMA '18*. 1–8. https://doi.org/10.23919/TMA.2018.8506558
[4] Alfred V. Aho and Margaret J. Corasick. 1975. Efficient String Matching: An Aid to Bibliographic Search. *Commun. ACM* 18, 6 (June 1975), 333–340.
[5] Khaled Al-Naami, Swarup Chandra, Ahmad Mustafa, Latifur Khan, Zhiqiang Lin, Kevin Hamlen, and Bhavani Thuraisingham. 2016. Adaptive Encrypted Traffic Fingerprinting with Bi-directional Dependence. In *ACSAC '16*. ACM, New York, NY, USA, 177–188. https://doi.org/10.1145/2991079.2991123
[6] Hasan Faik Alan and Jasleen Kaur. 2016. Can Android Applications Be Identified Using Only TCP/IP Headers of Their Launch Time Traffic?. In *WiSec '16*. ACM, New York, NY, USA, 61–66. https://doi.org/10.1145/2939918.2939929
[7] and Z. Chen, L. Zhang, Q. Yan, B. Yang, and L. Peng and. 2016. TrafficAV: An effective and explainable detection of mobile malware behavior using network traffic. In *IWQoS '16*. 1–6. https://doi.org/10.1109/IWQoS.2016.7590446
[8] Christian Borgelt. 2012. Frequent item set mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2, 6 (2012), 437–456. https://doi.org/10.1002/widm.1074
[9] Dhruba Borthakur et al. 2008. HDFS architecture guide. *Hadoop Apache Project* 53 (2008), 1–13.
[10] Y. Chen, J. Liu, and Y. Cui. 2016. Inter-player Delay Optimization in Multiplayer Cloud Gaming. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. 702–709. https://doi.org/10.1109/CLOUD.2016.0098
[11] Z. Chen, B. Yu, Y. Zhang, J. Zhang, and J. Xu. 2016. Automatic Mobile Application Traffic Identification by Convolutional Neural Networks. In *2016 IEEE Trustcom/BigDataSE/ISPA*. 301–307. https://doi.org/10.1109/TrustCom.2016.0077
[12] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms*. MIT press.
[13] A. Dainotti, A. Pescape, and K. C. Claffy. 2012. Issues and future directions in traffic classification. *IEEE Network* 26, 1 (2012), 35–40. https://doi.org/10.1109/MNET.2012.6135854
[14] Y. Fu, H. Xiong, X. Lu, J. Yang, and C. Chen. 2016. Service Usage Classification with Encrypted Internet Traffic in Mobile Messaging Apps. *IEEE Transactions on Mobile Computing* 15, 11 (Nov 2016), 2851–2864. https://doi.org/10.1109/TMC.2016.2516020
[15] Yubo Jiang, Xin Du, and Tao Jin. 2019. Using combined network information to predict mobile application usage. *Physica A: Statistical Mechanics and its Applications* 515 (2019), 430 – 439. https://doi.org/10.1016/j.physa.2018.09.135
[16] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I. Hong. 2018. Why Are They Collecting My Data?: Inferring the Purposes of Network Traffic in Mobile Apps. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 4, Article 173 (Dec. 2018), 27 pages. https://doi.org/10.1145/3287051
[17] J. Kuo, H. Ruan, C. Chan, and C. Lei. 2017. Investigation of mobile App behaviors, from the aspect of real world mobile backend system. In *AEECT '17*. 1–6. https://doi.org/10.1109/AEECT.2017.8257762
[18] Stanislav Miskovic, Gene Moo Lee, Yong Liao, and Mario Baldi. [n.d.]. AppPrint: Automatic Fingerprinting of Mobile Applications in Network Traffic. In *PAM '15*.
[19] G. Ranjan, A. Tongaonkar, and R. Torres. 2016. Approximate matching of persistent LExicon using search-engines for classifying Mobile app traffic. In *Infocom '16*. 1–9. https://doi.org/10.1109/INFOCOM.2016.7524386
[20] Abbas Razaghpanah, Arian Akhavan Niaki, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Johanna Amann, and Phillipa Gill. 2017. Studying TLS Usage in Android Apps. In *CoNEXT '17*. ACM, New York, NY, USA, 350–362. https://doi.org/10.1145/3143361.3143400
[21] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. 2016. ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic. In *MobiSys '16*. 361–374. https://doi.org/10.1145/2906388.2906392
[22] Brendan Saltaformaggio, Hongjun Choi, Kristen Johnson, Yonghwi Kwon, Qi Zhang, Xiangyu Zhang, Dongyan Xu, and John Qian. 2016. Eavesdropping on Fine-Grained User Activities Within Smartphone Apps Over Encrypted Network Traffic. In *WOOT '16*. USENIX Association, Austin, TX.
[23] Amedeo Sapio, Yong Liao, Mario Baldi, Gyan Ranjan, Fulvio Risso, Alok Tongaonkar, Ruben Torres, and Antonio Nucci. 2014. Per-user Policy Enforcement on Mobile Apps Through Network Functions Virtualization. In *MobiArch '14*. ACM, New York, NY, USA, 37–42. https://doi.org/10.1145/2645892.2645896
[24] Alexander Schrijver. 1998. *Theory of linear and integer programming*. John Wiley & Sons.
[25] M. Shen, M. Wei, L. Zhu, and M. Wang. 2017. Classification of Encrypted Traffic With Second-Order Markov Chains and Application Attribute Bigrams. *IEEE Transactions on Information Forensics and Security* 12, 8 (Aug 2017), 1830–1843. https://doi.org/10.1109/TIFS.2017.2692682
[26] H.W. Sorenson. 1980. *Parameter estimation: principles and problems*. M. Dekker.
[27] Tim Stöber, Mario Frank, Jens Schmitt, and Ivan Martinovic. 2013. Who Do You Sync You Are?: Smartphone Fingerprinting via Application Behaviour. In *WiSec '13*. ACM, New York, NY, USA, 7–12. https://doi.org/10.1145/2462096.2462099
[28] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic. 2016. AppScanner: Automatic Fingerprinting of Smartphone Apps from Encrypted Network Traffic. In *EuroS&P '16*. 439–454. https://doi.org/10.1109/EuroSP.2016.40
[29] A. Tongaonkar. 2016. A Look at the Mobile App Identification Landscape. *IEEE Internet Computing* 20, 4 (July 2016), 9–15. https://doi.org/10.1109/MIC.2016.77
[30] Alok Tongaonkar, Shuaifu Dai, Antonio Nucci, and Dawn Song. 2013. Understanding Mobile App Usage Patterns Using In-App Advertisements. In *PAM '13*, Matthew Roughan and Rocky Chang (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 63–72.
[31] Q. Wang, A. Yahyavi, B. Kemme, and W. He. 2015. I know what you did on your smartphone: Inferring app usage over encrypted data traffic. In *CNS '15*. 433–441. https://doi.org/10.1109/CNS.2015.7346855
[32] Shanshan Wang, Zhenxiang Chen, Qiben Yan, Ke Ji, Lin Wang, Bo Yang, and Mauro Conti. 2018. Deep and Broad Learning Based Detection of Android Malware via Network Traffic. In *IWQoS '18*. 1–6.
[33] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti. 2018. Detecting Android Malware Leveraging Text Semantics of Network Flows. *IEEE Transactions on Information Forensics and Security* 13, 5 (May 2018), 1096–1109. https://doi.org/10.1109/TIFS.2017.2771228
[34] W. Wang and J. Bickford. 2016. WhatApp: Modeling mobile applications by domain names. In *WiMob '16*. 1–10. https://doi.org/10.1109/WiMOB.2016.7763253
[35] Xuetao Wei, Nicholas C. Valler, Harsha V. Madhyastha, Iulian Neamtiu, and Michalis Faloutsos. 2017. Characterizing the behavior of handheld devices and its implications. *Computer Networks* 114 (2017), 1 – 12. https://doi.org/10.1016/j.comnet.2017.01.003
[36] Qiang Xu, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Jeffrey Pang, and Shobha Venkataraman. 2011. Identifying Diverse Usage Behaviors of Smartphone Apps. In *IMC '11*. ACM, New York, NY, USA, 329–344. https://doi.org/10.1145/2068816.2068847
[37] Q. Xu, Y. Liao, S. Miskovic, Z. M. Mao, M. Baldi, A. Nucci, and T. Andrews. 2015. Automatic generation of mobile app signatures from traffic observations. In *Infocom '15*. 1481–1489. https://doi.org/10.1109/INFOCOM.2015.7218526
[38] Hongyi Yao, Gyan Ranjan, Alok Tongaonkar, Yong Liao, and Zhuoqing Morley Mao. 2015. SAMPLES: Self Adaptive Mining of Persistent LExical Snippets for Classifying Mobile Application Traffic. In *MobiCom '15*. 439–451. https://doi.org/10.1145/2789168.2790097
[39] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.
[40] L. Zhang, F. Wang, and J. Liu. 2015. Improve Quality of Experience for Mobile Instant Video Clip Sharing. In *ICDCS '15*. 780–781. https://doi.org/10.1109/ICDCS.2015.106