Kaleido: Network Traffic Attribution using Multifaceted Footprinting

Ting Wang Fei Wang Reiner Sailer Douglas Schales
IBM Research

Abstract

Network traffic attribution, namely, inferring users responsible for activities observed on network interfaces, is one fundamental yet challenging task in network security forensics. Compared with other user-system interaction records, network traces are inherently coarse-grained, context-sensitive, and detached from user ends. This paper presents KALEIDO, a new network traffic attribution tool with a series of key features: a) it adopts a new class of inductive discriminant models to capture user- and context-specific patterns ("*footprints*") from different aspects of network traffic; b) it applies efficient learning methods to extracting and aggregating such footprints from noisy historical traces; c) with the help of novel indexing structures, it is able to perform efficient, runtime traffic attribution over high-volume network traces. The efficacy of KALEIDO is evaluated with extensive experimental studies using the real network traces collected over three months in a large enterprise network.

1 Introduction

Many of the vulnerabilities of today's network systems, such as denial-of-service attacks, IP address spoofing, and spamming, all root at the untraceable nature of network traffic. It is well recognized [1, 2] that in general without instrumental support it is extremely difficult to identify users responsible for observed network activities, due to the unique characteristics of network traces, including:

- In contrast of other user-system interaction records (e.g., mouse movement [20] and keyboard stroke [3]), network traces are inherently more coarse-grained and detached further from user ends. For instance, one network flow (i.e., *Netflow*) record may correspond to a multitude of keyboard strokes.
- Typically users' network activities vary significantly with contexts (e.g., time and location) [9], leading to complicated, context-sensitive patterns in network traces. As an example, in enterprise networks, users may demonstrate temporal patterns in performing activities such as emailing, browsing corporate websites, and accessing databases.
- Network traces often only remotely reflect the mixture behavior of both user and system. For instance, a significant amount of network traffic is generated by system update and maintenance.

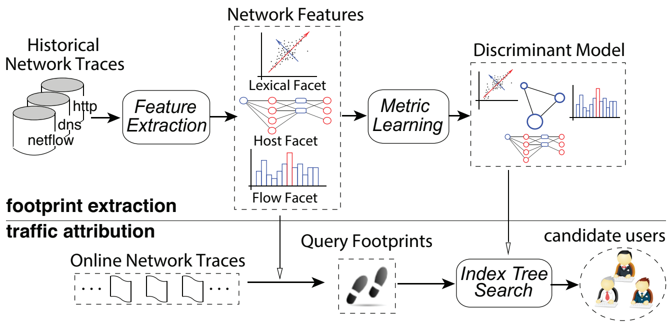

Figure 1: System architecture of KALEIDO.

This work targets an alternative objective: *building a practical tool capable of attributing a collection of network traces (associated with an unknown user) during a given time window to the most likely user(s) from a predefined pool (e.g., all the users of an enterprise network).* Although this result may not directly pinpoint the responsible user, it can be of great value for network security forensics. For example, an anomaly is raised if the suggested candidates are considered highly improbable with respect to other side information (e.g., the credentials of operating users).

However, building such tools requires addressing a number of non-trivial challenges, including: (i) how to extract user- and context-specific traffic patterns (which we refer to as users' "*footprints*"[1]) from noisy historical traces? (ii) how to aggregate possibly weak footprints from different aspects of traffic measures (which we refer to as "*facets*", e.g., visited URLs, traffic flow size, traffic elapse time) to improve the accuracy of attribution? (iii) how to scale up the solution to large user pools and high-volume network traces?

This paper presents KALEIDO, a new network traffic attribution tool, which meets all these requirements. As shown in Figure 1, KALEIDO comprises two main phases, *footprint extraction* and *traffic attribution*. For the first phase, we formalize it as learning a class of inductive discriminant functions for user pairs from historical traces. Concretely we propose a large margin formalization that optimally integrates information from multiple facets of network traces. While difficult to solve in the primal space, this problem is efficiently solvable in its dual

[1]This concept contrasts "*fingerprint*" which refers to patterns that uniquely identify users for practical purposes.

facet	example	trace
uniform resource locator (URL)	*http://www.example .com/sample.html*	HTTP request
domain name	*example.net*	DNS query
host country	*US*	DNS query
traffic flow size	*2,048 packets*	Netflow record
flow elapse time	*1,024 milliseconds*	Netflow record

Table 1: Example facets of network traces.

form, which makes Kaleido practical for large-scale settings. For the second phase, we propose novel indexing structures for the extracted footprints, and show how to perform efficient runtime footprint matching over high-volume network traces. To our best knowledge, this is the first proposal to use multifaceted footprints to perform scalable network traffic attribution.

We evaluate Kaleido by extensive empirical studies on the real network traces of 1,242 users collected over 3 months in a large enterprise network. We show that by examining the network traces of unknown users for merely a short time window (e.g., $2 \sim 4$ hours), Kaleido is able to identify the responsible users with over 92% accuracy, while it takes less than 0.5 second to do so on a machine with modest hardware configuration.

The rest of the paper proceeds as follows. Section 2 formalizes the problem of network traffic attribution. Section 3 and 4 describe in detail the model and algorithm design of the main components of Kaleido. Section 5 addresses the detailed issues of implementing Kaleido. An empirical evaluation of Kaleido is presented in Section 6. Section 7 surveys relevant literature. The paper is concluded in Section 8.

2 Formalization

This section introduces fundamental concepts and notations used throughout the paper and formalizes the problem of network traffic attribution.

We consider a variety of network traces including: *DNS query*, *HTTP request header*, and *Netflow measure*. Despite their disparate forms, we unify the processing of these traces by introducing the concept of *facet*.

Definition 1 (facet). *A facet is a specific measurement dimension of network traces, e.g., visited url, traffic flow size, and traffic elapse time.*

Table 1 lists a set of example facets and their associated network traces. Note that certain traces (e.g., Netflow measure) may have multiple facets. For simplicity of presentation, we treat each facet independently and discuss in Section 5 the extension of leveraging the relationships between different facets.

We assume that the traces with respect to a specific facet consist of a set of *measure records.*

Definition 2 (measure record). *A measure record is a tuple ⟨facet, context, value⟩, representing respectively the facet to which this measure belongs, the context under which this measure is taken (e.g., time, location, operating system platform), and the value of this measure (e.g., the size of a network flow).*

We further project the raw measure records to the *feature space.* For example, we may divide the range of traffic flow size into a set of disjoint buckets and consider each bucket as a feature. The details of feature selection for concrete network traces will be discussed in Section 5.

We are now ready to introduce the problem of *network traffic attribution*:

Definition 3 (network traffic attribution). *Based on a set of measure records relevant to an unknown user from different facets over a given time window, inferring the most likely responsible user(s) from a predefined pool (e.g., all registered users of an enterprise network).*

To this end, we intend to extract user- and context-specific patterns from network traces and leverage such patterns to differentiate each user from the rest. Unfortunately, as introduced in Section 1, network traces are inherently coarse-grained and detached from user ends, which makes such patterns often not distinguishing enough to pinpoint individual users. We thus refer to such patterns as "footprints".

The rationale of Kaleido is to fuse the information from different facets by integrating multiple footprints, thereby leading to improved attribution accuracy. Note that this is usually feasible in today's enterprise networks wherein multiple traffic monitoring tools are deployed (e.g., *PacketTrap*).

3 Footprint Extraction

In the footprint extraction phase, for each pair of users in the pool, Kaleido learns an inductive discriminant function from the multifaceted network measures in historical traces (training data). Next we formalize this learning problem and present efficient solutions.

3.1 Discriminant function. Assuming a predefined pool of U users, we intend to learn a binary *inductive discriminant function* f that differentiates each pair of users based on their characterization from K different facets of network traces. Formally, let $x^{(k)}, y^{(k)} \in \mathbb{R}^{d_k}$ denote the feature vectors of two users in the k-th facet, where d_k is the number of features. Let x, y be the feature vectors aggregated over the K facets. Then Kaleido aims to learn a function $f : \mathbb{R}^{\sum_{k=1}^{K} d_k} \times \mathbb{R}^{\sum_{k=1}^{K} d_k} \rightarrow \{+1, -1\}$, such that

$$f(x, y) = \begin{cases} 1, & \text{if } x,\, y \text{ correspond to a same user} \\ -1, & \text{if } x,\, y \text{ correspond to different users} \end{cases}$$

Its "inductive" nature implies that once obtained, f can be applied to match new users not in the training data.

Concretely we design a general linearly aggregated quadratic discriminant form for f:

$$f(x, y) = \sum_{k=1}^{K} \mu_k \left[\frac{1}{2} (z^{(k)})^\top Q^{(k)} z^{(k)} + (r^{(k)})^\top z^{(k)} \right] + c$$

where $z^{(k)} = \begin{bmatrix} x^{(k)} \\ y^{(k)} \end{bmatrix}$ and $\{Q^{(k)}, r^{(k)}, \mu_k\}_{k=1}^K$ is a set of parameters: $Q^{(k)} \in \mathbb{R}^{2d_k \times 2d_k}$ is a real symmetric matrix (not necessarily positive semidefinite), $r^{(k)} \in \mathbb{R}^{2d_k}$ is a column vector, and $\{\mu_k\}_{k=1}^K$ lie in a simplex, that is, $\sum_{k=1}^K \mu_k = 1, \mu_k \geqslant 0$ $(k = 1, 2, \cdots, K)$.

Compared with the conventional Mahalanobis distance function [5, 6, 17, 18], this general second-order discriminant function takes into consideration the variations in the network traces of different users (e.g., background traffic) and focuses on user-intrinsic patterns.

3.2 Model details.

Given the symmetry of f, i.e., $f(x, y) = f(y, x)$, $Q^{(k)}, r^{(k)}$ take the following symmetric block-wise form:

$$Q^{(k)} = \begin{bmatrix} A^{(k)} & B^{(k)} \\ (B^{(k)})^\top & A^{(k)} \end{bmatrix} \quad r^{(k)} = \begin{bmatrix} b^{(k)} \\ b^{(k)} \end{bmatrix}$$

Then we can rewrite $f(x, y)$ as:

$$\begin{aligned} (3.1)\; f(x,y) &= \sum_k \mu_k \left[\frac{1}{2}(x^{(k)})^\top A^{(k)} x^{(k)} \right. \\ &\quad + \frac{1}{2}(y^{(k)})^\top A^{(k)} y^{(k)} + (x^{(k)})^\top B^{(k)} y^{(k)} \\ &\quad \left. + (b^{(k)})^\top x^{(k)} + (b^{(k)})^\top y^{(k)} \right] + c \end{aligned}$$

Here we focus on the supervised setting. We assume that the training data is given as $\mathcal{P} = \{(x_i, y_i)\}$ where (x_i, y_i) is the feature vectors of a pair of users in the user pool. For each (x_i, y_i), a binary label l_i is provided: $l_i = +1$ indicates that x_i and y_i correspond to a same user; and $l_i = -1$ otherwise. Let $\mathcal{P}_+$ and $\mathcal{P}_-$ be the set of positive and negative pairs, respectively. We propose the following large margin formulation for the footprint extraction problem:

$$\min \frac{1}{2} \sum_k \mu_k \left(\|A^{(k)}\|_F^2 + \|B^{(k)}\|_F^2 + \|b^{(k)}\|_2^2 \right) + \lambda \sum_{i \in \mathcal{P}} \xi_i$$
$$\text{s.t.} \begin{cases} f(x_i, y_i) \geqslant +1 - \xi_i & \forall i \in \mathcal{P}_+ \\ f(x_i, y_i) \leqslant -1 + \xi_i & \forall i \in \mathcal{P}_- \\ \xi_i \geqslant 0 & \forall i \in \mathcal{P} \end{cases}$$

where $||\cdot||_F$ and $\|\cdot\|_2$ denote the matrix Frobenius norm and vector Euclidean norm, respectively. This formulation intuitively maximizes the generalization ability of f while discriminating the positive and negative pairs to the maximum extent.

3.3 Learning method.

For notational convenience, we introduce the following vectorized representation:

$$\rho^{(k)} = \mu_k \left[(\text{vec}(A^{(k)}))^\top, (\text{vec}(B^{(k)}))^\top, (b^{(k)})^\top \right]^\top$$

$$\phi_i^{(k)} = \frac{1}{2} \begin{bmatrix} \text{vec}(x_i^{(k)}(x_i^{(k)})^\top + y_i^{(k)}(y_i^{(k)})^\top) \\ \text{vec}(x_i^{(k)}(y_i^{(k)})^\top + y_i^{(k)}(x_i^{(k)})^\top) \\ 2(x_i^{(k)} + y_i^{(k)}) \end{bmatrix}$$

where vec$(\cdot)$ denotes the vectorization of a matrix. We can thus reformulate the optimization problem as:

$$\min \frac{1}{2} \sum_k \frac{1}{\mu_k} \|\rho^{(k)}\|_2^2 + \lambda \sum_{i \in \mathcal{P}} \xi_i$$

$$\text{s.t.} \begin{cases} l_i \left[\sum_{k=1}^K \left\langle \rho^{(k)}, \phi_i^{(k)} \right\rangle + c \right] \geqslant 1 - \xi_i & \forall i \in \mathcal{P} \\ \xi_i \geqslant 0 & \forall i \in \mathcal{P} \\ \sum_k \mu_k = 1, \; \mu_k \geqslant 0 & k = 1, 2, \cdots, K \end{cases}$$

While directly solving this problem is expensive due to the high dimensionality of $\rho^{(k)}$ and $\phi^{(k)}$ $(2d_k^2 + d_k)$, its dual form is efficiently solvable using the *reduced gradient descent* method [14]. More specifically, we can rewrite the above problem as:

$$\min_\mu \; J(\mu) \quad \text{s.t.} \sum_k \mu_k = 1, \; \mu_k \geqslant 0, \; k = 1, 2, \cdots, K$$

where $J(\mu)$ is defined as:

$$J(\mu) = \begin{cases} \inf_{\rho, \xi, c} & \frac{1}{2} \sum_k \frac{1}{\mu_k} ||\rho^{(k)}||_2^2 + \lambda \sum_{i \in \mathcal{P}} \xi_i \\ \quad \text{s.t.} & l_i \left[\sum_k \langle \rho^{(k)}, \phi_i^{(k)} \rangle + c \right] \geqslant 1 - \xi_i \\ & \xi_i \geqslant 0, \; \forall i \in \mathcal{P} \end{cases}$$

It is noted that this objective function is essentially an optimal SVM value. Assuming that J is differentiable, we solve this problem in its dual form using reduced gradient descent methods, with details shown in Algorithm 1, where $M^{(k)}$ is the data inner product matrix (kernel matrix) of the k-th facet. To ensure the equality and non-negativity constraints of μ, the gradient direction ϵ is defined as (where δ is the index of the largest component of μ):

$$\epsilon_k = \begin{cases} 0 & \text{if } \mu_k = 0 \text{ and } \frac{\partial J(\mu)}{\partial \mu_k} > \frac{\partial J(\mu)}{\partial \mu_\delta} \\ -\frac{\partial J(\mu)}{\partial \mu_k} + \frac{\partial J(\mu)}{\partial \mu_\delta} & \text{if } \mu_k > 0 \text{ and } k \neq \delta \\ \sum\limits_{\tau \neq \delta, \mu_\tau > 0} \left(\frac{\partial J(\mu)}{\partial \mu_\tau} - \frac{\partial J(\mu)}{\partial \mu_\delta} \right) & \text{if } k = \delta \end{cases}$$

Algorithm 1: minimize $J(\mu)$ with reduced gradient descent

```
// initialization
$\mu_k \leftarrow \frac{1}{K}$ for $k = 1, 2, \cdots, K$;
while stopping criterion not met do
    compute $J(\mu)$ using SVM solver with kernel matrix $M = \sum_k \mu_k M^{(k)}$;
    compute $\frac{\partial J(\mu)}{\partial \mu_k}$ for $k = 1, 2, \cdots, K$ and descent direction $\epsilon$;
    $\delta \leftarrow \arg\max_k \mu_k$, $J^+ \leftarrow 0$, $\mu^+ \leftarrow \mu$, $\epsilon^+ \leftarrow \epsilon$;
    // descent direction update
    while $J^+ < J(\mu)$ do
        $\mu \leftarrow \mu^+$, $\epsilon \leftarrow \epsilon^+$;
        $\tau \leftarrow \arg\min_{\{k|\epsilon_k<0\}} -\frac{\mu_k}{\epsilon_k}$, $\gamma_{max} \leftarrow -\frac{\mu_\tau}{\epsilon_\tau}$;
        $\mu^+ \leftarrow \mu + \gamma_{max}\epsilon$, $\epsilon_\delta^+ \leftarrow \epsilon_\delta - \epsilon_\tau$, $\epsilon_\tau^+ \leftarrow 0$;
        compute $J^+$ using an SVM solver with kernel matrix $M = \sum_k \mu_k^+ M^{(k)}$;
    line search along $\epsilon$ for $\gamma \in [0, \gamma_{max}]$;
    $\mu \leftarrow \mu + \gamma\epsilon$;
```

4 Traffic Attribution

In this phase, a set of footprints (relevant to a certain unknown user) from multiple facets are submitted as a query to Kaleido. By deriving the similarity between the query footprints and the footprints of each user in the user pool, Kaleido identifies the most similar users as the candidates. Next we present novel indexing structures to scale up this matching process to handle high-volume incoming queries.

4.1 Problem reformulation. For simplicity of presentation, we assume that all the measure records have been transformed into feature vectors as in Section 3. Let x be a feature vector in the training data $\mathcal{P}$ and y be the query feature vector. The footprint matching problem is essentially equivalent to the search problem: $\arg\max_{x \in \mathcal{P}} f(x, y)$.

A straightforward solution is to compute the discriminant function $f(\cdot)$ for each feature vector $x \in \mathcal{P}$ with respect to y, which however is prohibitively expensive for large datasets. Concretely, assuming m feature vectors corresponding to n different users ($m \gg n$) in the training dataset $\mathcal{P}$ (that is, each user may have multiple feature vectors under different contexts, details given in Section 5), such linear search approach features complexity of $O(m \sum_k d_k^2)$.

Next we introduce a novel tree-based branch-and-bound approach which reduces the complexity of footprint matching to $O(\sum_k d_k^2 + \log m \sum_k d_k)$ in most cases, thereby practical for runtime execution.

We first rewrite the discriminant function $f(x, y)$ of Eq. (3.1) in the following form:

$$\begin{aligned} f(x,y) &= \underbrace{\sum_k \mu_k \left[\frac{1}{2}(x^{(k)})^\top A^{(k)} x^{(k)} + (b^{(k)})^\top x^{(k)}\right] + c}_{\alpha(x)} \\ &\quad + \underbrace{\frac{1}{2}\sum_k \mu_k (y^{(k)})^\top A^{(k)} y^{(k)}}_{\beta(y)} \\ &\quad + \sum_k \mu_k \left[(x^{(k)})^\top A^{(k)} + (b^{(k)})^\top\right] y^{(k)} \end{aligned}$$

It is noticed that the first term of the right side of the equation ($\alpha(x)$) only depends on x and thus can be precomputed, while the second term ($\beta(y)$) only depends on y and is computable on the fly in time $O(\sum_k d_k^2)$. We then construct the following two vectors:

$$(4.2)\quad x^* = \begin{bmatrix} (A^{(1)})^\top x^{(1)} + b^{(1)} \\ \vdots \\ (A^{(K)})^\top x^{(K)} + b^{(K)} \\ 1 \\ \alpha(x) \end{bmatrix} \quad y^* = \begin{bmatrix} \mu_1 y^{(1)} \\ \vdots \\ \mu_K y^{(K)} \\ \beta(y) \\ 1 \end{bmatrix}$$

It is clear that the computation of $f(x, y)$ is now translated into the computation of inner product of x^* and y^*. Therefore finding the most matching footprint is essentially equivalent to searching for the maximum inner product for y^* with respect to a set of reference vectors $\{x^*\}$, more formally

$$\arg\max_{x \in \mathcal{P}} f(x, y) \equiv \arg\max_{x \in \mathcal{P}} \langle x^*, y^* \rangle$$

Algorithm 2: find most similar footprints (query y, training data $\mathcal{P}$)

```
// precomputation
foreach feature vector $x \in \mathcal{P}$ do
    compute $x^*$ according to Eq. (4.2);
construct ball tree index $\mathcal{T}$ on $\{x^*\}$;
// tree-based search
compute $y^*$ from $y$ using Eq. (4.2);
search $\mathcal{T}$ for $\arg\max_x \langle x^*, y^* \rangle$;
```

4.2 Branch-and-bound search. We build ball tree-based indexes [13] to facilitate such search tasks.

Ball trees are binary space-partitioning structures that divide a set of points hierarchically into possibly overlapping hyper-spheres. Every tree node represents a subset of points and is indexed by a center and a sphere enclosing all the points in the subset. Every non-leaf node is further partitioned into two disjoint subsets, represented by two child nodes. The search of the maximum inner product for query y^* with respect to a point set $\{x^*\}$ is performed in a depth-first branch-and-bound manner: beginning at the root, traversing down the tree node with the maximum inner product between y^* and any potential vectors in the node, and performing linear search if reaching a leaf. Particularly we use the bound in [15] to estimate the potential maximum inner product for every tree node.

Note that the extension to the case of retrieving top k ($k > 1$) best matches is straightforward: (1) maintain a list of current top k maximum inner products $\{\theta\}$; (2) skip a tree node only if its bound is below the smallest θ; (3) update the list if a newly found vector with inner product larger than the smallest θ.

It is also noted that this framework can be fully parallelized by distributing the footprints to different machines, executing the matching algorithms independently, and finally aggregating the matching results. We omit the details here due to the space limitations.

5 Implementation

This section addresses the detailed issues of implementing Kaleido. We start with describing the datasets used in the training phase.

5.1 Data collection. We use a large corpus of real network traces collected from a large-scale enterprise network, which corresponds to the network activities of 6,791 different MAC addresses and 1,242 distinct users over 3 months (from May 1st to July 31st, 2011). The dataset comprises three main parts, HTTP request headers, Netflow measures, and DNS queries. Note that while some other types of network traces (particularly content relevant traces, e.g., HTTP payloads) provide more detailed recording of the behavior of network users, their availability is often questionable in many settings (e.g., HTTPs); meanwhile we intend our methods to be generally applicable. Therefore here we focus on the most commonly available traces.

lexical-based		host-based		flow-based	
facet name	**feature #**	**facet name**	**feature #**	**facet name**	**feature #**
domain name	664,963	IP prefix	113,496	in/outbound flow #	N/A
path (tokenized)	1,523,482	geographic	59	flow size (logarithmic)	32
				traffic elapse time (logarithmic)	12

Table 2: Feature breakdown for one month of network traces (in/outbound flow # is time-window dependent).

Additionally for the training phase we have access to two sets of mappings. The first one is the Dynamic Host Configuration Protocol (DHCP) logs that map each IP address at a given timestamp to its corresponding MAC address. The second one is the user authentication logs that map each MAC address at a given timestamp to the logon credential of the responsible user. With the help of these two mappings (IP → MAC → user's credential), we are able to attribute the network traces to the responsible users. Note that we only use this information in the training phase and as the ground truth for evaluation.

5.2 Feature extraction. From the network traces, we extract a set of *lexical*-, *host*-, and *flow*-based facets, which are listed in Table 2. We next briefly describe these facets and the motivation behind including them for network traffic attribution tasks.

Lexical-based facets: These facets capture the behavior of network users in terms of "what" types of websites they have visited and "how" they have interacted with these websites. We extract such facets from the URLs embedded in the HTTP requests. For a given URL (e.g., 'www.research.ibm.com/Search/?q=smarter+planet'), we consider that its *domain name* (e.g., 'research.ibm.com') lexically encodes the classification of the website, while the remainder part of the URL (path) further details the input/output information (especially for dynamically generated pages, e.g., 'Search/?q=smarter+planet').

Instead of directly using domain names and paths (which may lead to very sparse feature spaces), we transform them into more consumable features. Concretely, by feeding to domain name categorization services (e.g., Cobian: www.iss.net), we classify domain names into a set of predefined categories (e.g., 'www.research.ibm.com' is categorized as 'IT research/hardware/software/service'). For path values, we tokenize each path using '/', '?', '.', '=', '_' and '-' as delimiters and adopt a bag-of-word representation of the tokens.

Host-based facets: These facets describe the infrastructure properties of the websites which users have visited, including "where" the websites are hosted, "who" owns them, and "how" they are managed. In particular, *IP prefix* approximates the Internet Service Provider (ISP) that hosts the website, *Geographic* information gives location information of the host, and *WHOIS* includes registrant and registrar information of the host.

Flow-based facets: These facets measure the statistics of Web traffic on the basis of network flows. More

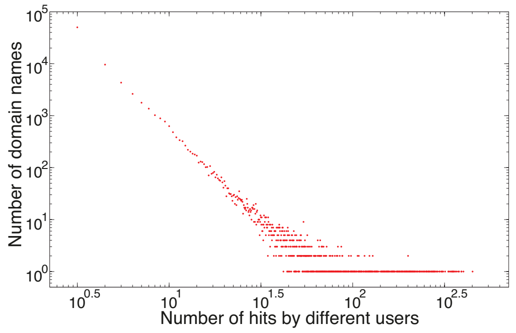

Figure 2: Distribution of hit numbers of domain names.

specifically, we differentiate inbound (from website to user) and outbound (from user to website) network traffic. For traffic in both directions, we count the number of flows in the specified time window, and collect the distribution of flow size, the elapse time of each flow, and the interval time between flows. Particularly we discretize the flow size (number of packets) and elapse time (seconds) into buckets which increases in a geometric manner (i.e., 2^0, 2^1, 2^2, . . .).

Table 2 lists the cumulative number of distinct feature values of each facet in the training data over one month (May 2011). It is noted that under this setting the network measures of each user in every facet can be succinctly described by a histogram. We therefore concatenate the facets in each category to form a single facet, which in the following we use "flow facet", "host facet", and "lexical facet" to refer to.

One may have the question that in enterprise networks users may perform fairly similar network activities (e.g., emailing, browsing corporate websites, accessing databases), would these features be effective enough to differentiate users? Here we use the facet of domain name as an example to show the distinguishing power of these facets. Figure. 2 illustrates the distribution of domain names according to their hit numbers by different users. The power law-like distribution shows clearly that in enterprise network environments, different users may demonstrate discernible patterns even in the most commonly available network measures.

5.3 Context awareness. The basic footprint model is defined in a context-agnostic manner, which ignores the strong correlation between the user behavior and its context information, e.g., a user's network activities on weekdays tend to be different from that on weekends.

Here we incorporate the rich context information associated with users' network traces. While our framework is easily extensible to accommodate other types

context	definition
day of week (DOW)	weekday or weekend
time of day (TOD)	morning or afternoon
system platform (SP)	user MAC address user operating system

Table 3: Context information used by Kaleido.

of context information, in this work we focus on three types of contexts (listed in Table 3): *day of week* (DOW), *time of day* (TOD), and *system platform* (SP). Specifically, DOW differentiates weekdays and weekends, TOD represent the recording time of network measures, and SP includes the information such as the MAC address (available in DHCP log) and the operating system (available in the *User-Agent* field of HTTP request header) of the user device.

We organize the training data into *shards*, each corresponding to a specific context setting. For example, we may have one shard for each combination of the following Cartesian product: {morning, afternoon} × {weekday, weekend} × {Windows, *nix}. We generate a positive or negative case for a pair of feature vectors only if they correspond to the same context.

As we will see in Section 6, the introduction of context awareness not only improves the accuracy of network traffic attribution, but also reduces the complexity of model learning and maintenance because an attribution model is trained specifically for each shard and used exclusively on the testing cases in that shard.

5.4 Additional information. In the above discussion, we consider each user, facet, and context in a separate manner. However, in reality there may exist semantically rich correlation between these aspects. For example, users may exist in a managerial hierarchy and users with similar employment positions tend to demonstrate similar network behavior; each website may have specific flow size distribution; and a user's Web activities may vary from day to night but may show strong regularity during daytime.

With such side information available, it is possible to incorporate user-, facet-, or context-correlation in the formalization of footprint model, e.g., using the context network model [11]. We consider such enhancement as one direction of our ongoing research.

6 Evaluation

This section presents an empirical analysis of Kaleido. We aim at answering the following questions: 1) How effective is Kaleido in identifying users responsible for observed network traces? 2) What advantages does the multifaceted footprinting model offer over alternative techniques? 3) Is Kaleido able to scale up to large user pools and high-rate network traces? 4) As the network traffic evolves over time, what training regimens should Kaleido follow to retain sufficient accuracy.

6.1 Evaluation setting. Among the network traces collected over 92 days, we use the traces of the first 46

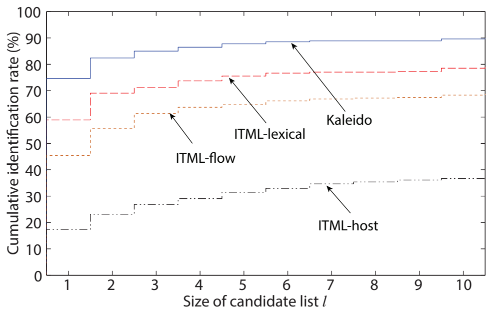

Figure 3: Accuracy of network user re-identification versus candidate list size l ($k = 5$ in k-NN classification, observation window length = 4 hours).

	$k = 1$	$k = 3$	$k = 5$	$k = 7$	$k = 9$
ITML-h	0.5773	0.6002	0.6176	0.6352	0.6470
ITML-f	0.6687	0.7086	0.7251	0.7413	0.7510
ITML-l	0.7843	0.8098	0.8231	0.8329	0.8368
Kaleido	**0.7927**	**0.8240**	**0.8399**	**0.8449**	**0.8421**

Table 4: Hand-Till AUC scores of traffic attribution models versus varying setting of k.

days to train the traffic attribution models and the rest as the testing set. We further divide the network traces into *shards*, each corresponding to one specific context setting (as defined in Section 5.3), e.g., {*weekday*, *afternoon*, *Windows*}. Note that an attribution model is trained specifically for each shard and used exclusively on the testing data in that shard.

We compare the performance of Kaleido against ITML [5], the state-of-the-art metric learning algorithm. As ITML is designed for single-facet metric learning, we thus implement three variants based on ITML, that uses the host-facet only, that uses both host- and flow-facet, and that uses all host-, flow- and lexical-facet, denoted by ITML-h, ITML-f, and ITML-l, respectively. For ITML-f and ITML-l, we concatenate the feature vectors from different facets and perform ITML over the concatenated feature vectors.

In the training phase, in each shard, we randomly select $0.1 \times n^2$ (n denotes the number of users) pairs of data points (each corresponding to the activities of one particular user in that shard during one day). The points belonging to a same user are constrained to be similar, and otherwise constrained to be dissimilar. For ITML particularly we initialize its Mahalanobis distance matrix using squared Euclidean distance.

We use Kaleido and ITML to construct k-nearest neighbor (k-NN) classifiers, which rank the likelihood of each user in the pool being responsible for a query (observed network traces relevant to an unknown user). The default length of observation time window length is 4 hours, and each time 1,000 queries are evaluated. All the experiments are conducted on a Linux box running 2.40 GHz Intel processors and 8 GB memory. All the results here are the average of 10 repeated runs.

6.2 Accuracy of traffic attribution. This set of experiments are designed to measure the effectiveness of Kaleido and ITML to re-identify responsible users using their behavioral patterns embedded in their generated network traces. Specifically we submit the network traces generated by "unknown" users as queries and let the traffic attribution tools suggest the most likely users responsible for the queries. Here we pick the top candidate suggested by each attribution method and compare the performance of different methods in re-identifying the true identities.

Table 4 lists the Hand-Till AUC scores [8] of different attribute models with respect to the setting of k in k-NN classifiers. The Hand-Till AUC score is an extension of AUC scores for binary classification to multi-class cases using pair-wise approximation. It is clear that: 1) the performance of different variants of ITML varies significantly. For example, under $k = 1$ the performance of ITML-h is close to random guess (with AUC score 0.5) while ITML-l achieves much higher accuracy. 2) Kaleido outperforms all the variants of ITML across all the settings of k. This can be intuitively explained by the advantage of multifaceted footprinting, which optimally combines the user-distinguishing power of each facet. Also from the performance of variants of ITML, we can notice the relative value of different facets in network traffic attribution, roughly in the order of lexical > flow > host. This is explained by the nature of these facets (see Section 5): host and flow information is typically more coarse-grained than lexical information. For example, one Internet Service Provider (ISP) may host multiple websites with different domain names, while different web pages may have fairly similar sizes.

Next, instead of focusing on the most likely identity, we allow attribution methods to return a list of candidates. The overall accuracy is measured by the cumulative probability that the responsible user appears in this candidate list. Figure 3 illustrates how the user re-identification rate of each method grows as the size of candidate list varies from 1 to 10 (average 287 users in each shard). It is noticed that combing multiple facets considerably boots the re-identification accuracy. For example, the first guess of Kaleido achieves about 75% accuracy, in comparison of about 59%, 45%, and 19% identification rates by variants of ITML. This initial result suggests that Kaleido is able to effectively capture user-specific and context-dependent behavioral patterns embedded in network traces.

An important question thus arises: how many evidences are necessary for Kaleido to achieve reliable traffic attribution accuracy, in other words, how complete should the observation of network traces be? We answer this question by evaluating the user re-identification accuracy of different models by changing the length of observation window. The results are plotted in Figure 4, in which we vary the observation window length from 1 hour to 6 hours. As expected, the user re-identification accuracy of all the models grow steadily as the window length increases, e.g., for Kaleido, from below 60% to above 75% with candidate list size fixed as 1 in the cases

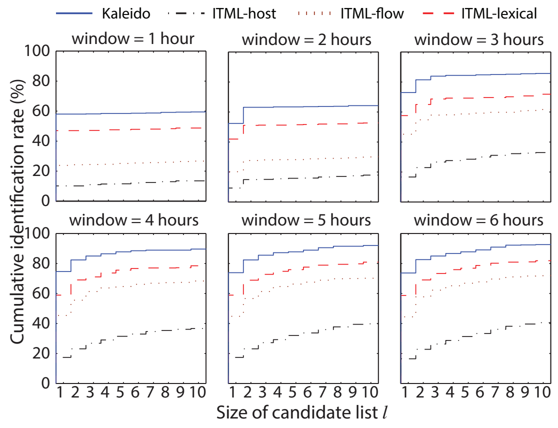

Figure 4: Accuracy of network user re-identification with respect to length of observation window ($k = 5$ in k-NN classification).

of 1-hour window and 6-hour window, respectively; however, the growth margin decreases gradually. It can be concluded that with very short observation windows (e.g., 3 hours), Kaleido is already able to achieve accuracy sufficient for practical use.

The use of such results however varies with the concrete applications. Next we show one concrete case in network security forensics.

6.3 Detection of unseen users.

One possible application of network traffic attribution tools is to detect anomaly of a Web user, i.e., deviation from his or her expected behavior. For example, an unauthorized user may use stolen credential to operate in place of the legitimate user. We examine the ability of Kaleido to detect these cases by measuring how often our approach would correctly detect an unseen user (e.g., a fraudulent user who has claimed to be another user).

	$l = 1$	$l = 2$	$l = 3$	$l = 4$	$l = 5$
ITML-h	0.6332	0.6362	0.6394	0.6405	0.6439
ITML-f	0.7126	0.7144	0.7040	0.6720	0.6170
ITML-l	0.7245	0.7369	0.7307	0.6803	0.6145
Kaleido	**0.8869**	**0.8906**	**0.8853**	**0.8792**	**0.8356**

Table 5: AUC scores of unseen-user detection versus candidate list size l ($k = 5$ in k-NN classification, observation window length = 4 hours).

In particular, we fix the size of the candidate list returned by each method and set a threshold on the cumulative classification probability of top-l candidates. If the cumulative probability of these candidates exceed the threshold, the network traces in the query are accepted as belonging to a legitimate user in the candidate list; otherwise, a fraudulent alert is raised.

We use the same setting as previous experiments. Further, for each set of experiments, we generate 500 positive (with responsible users in the training set) and 500 negative queries (with responsible users not in training set). As it is a binary classification problem, we use AUC score to measure the performance of different methods in this case, with results listed in Table 5. We see that overall Kaleido is the only algorithm to obtain the highest scores across all the settings of l. We also

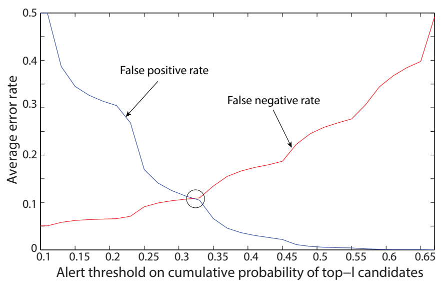

Figure 5: False positive and false negative rates versus setting of alert threshold (number of candidates $l = 3$, $k = 5$ in k-NN classification).

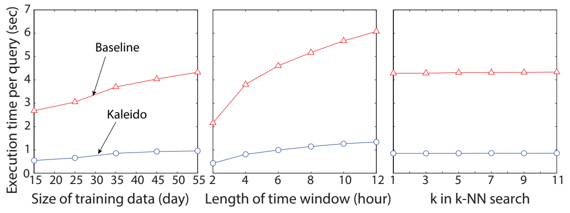

Figure 6: Efficiency of query answering versus size of training data, length of observation window and k in k-NN classification (default setting: training phase = 46 days, window length = 4 hours, $k = 5$).

observe that $l = 2, 3$ may yield the optimal performance of anomaly detection.

Another crucial parameter is the threshold on the cumulative classification probability of top-l candidates. We use two metrics to tune this parameter: *false positive* rate measures the probability that given network traces would be rejected though they are actually generated by legitimate users; and *false negative* rate measures the probability that given network traces of any unseen users would be accepted as legitimate traces.

The results are depicted in Figure 5. As we increase the alert threshold, the false positive rate drops almost precipitously while the false negative rate increases roughly linearly. These error rates intersect at an alert threshold approximately 0.32, where the false positive and false negative rates are both about 0.1, which suggests an optimal setting for the alert threshold.

6.4 Scalability of runtime operation. This set of experiments are designed to measure the scalability of runtime traffic attribution operation by Kaleido. In addition to Kaleido, we implemented a baseline approach that straightforwardly compares the footprints of query traces against that of training data.

A number of factors influence the runtime efficiency of Kaleido. The space limitations preclude the possibility of detailed analysis. Here we focus on three main factors, size of training data, length of observation window, and k in k-NN classification. Specifically, we evaluate the average processing time for each incoming query by changing one of these factors and keeping the rest fixed. Figure 6 shows the average execution time of Kaleido and the baseline method. As we can notice that with the help of novel indexing structure, Kaleido is order of magnitude faster than the baseline method, and is much less sensitive to the size of training data and the volume of query traces.

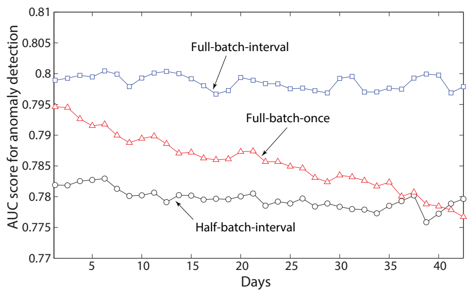

Figure 7: AUC score for anomaly detection of batch training and interval training algorithms ($l = 3$, $k = 5$, observation window length = 4 hours).

Also note that while all the experiments are conducted on a single PC, the footprint matching component of Kaleido is fully parallelizable, which makes it suitable for handling high-volume network traces.

6.5 Retraining regimen. In the last set of experiments, we intend to understand the impact of the temporal evolution of network traces on the performance of Kaleido and the best training regimen to retain sufficient accuracy.

For comparison purpose, we implemented three training schemes: *full-batch-once*, which trains once on Day 0 using the data of the past 30 days and uses that model for testing on all other days, *half-batch-interval*, which trains every 5 days using the data of the past 15 days, and *full-batch-interval*, which trains every 5 days with the data of the past 30 days. Figure 7 illustrates the AUC scores for anomaly detection using models trained by these schemes.

Clearly the performance of *full-batch-once* scheme deteriorates continuously as time evolves. This validates our assumption that users' network behavior demonstrates temporal evolution. Both *full-batch-interval* and *half-batch-interval* schemes address this issue by retraining the models on an interval basis, thereby maintaining steady performance across the testing phase of 45 days. Further, the *full-batch-interval* scheme outperforms *half-batch-interval* significantly as it exploits more training data. Therefore we conclude that an ideal training regimen requires both retraining frequency and training data volume, while the efficiency of our footprint learning algorithm is crucial for this purpose.

7 Related Work

Attributing user-system interaction traces to responsible users is one fundamental yet challenging task in security forensic analysis. Existing work has studied the problem in various contexts, including biometric traces (e.g., acoustic features [12] and facial features [7, 10]), user-device interaction traces (e.g., keyboard stroke [3], mouse movement [20], and even fill-in-bubble forms[4]), and system logs (e.g., console and SQL command traces

[16]). Unlike these user attribution tasks that take input directly from the user ends, attributing network traces are inherently more difficult for the traces are typically much more coarse-grained, context-dependent, and detached from user ends. To our best knowledge, this is the first proposal that studies the viability of large-scale network traffic attribution.

There are several other lines of work we build our techniques upon. Our footprint model is related to existing work in metric learning [7, 10, 19], which aims at learning a proper function to measure the distance between each pair of data objects x and y. The most popular choice is the Mahalanobis distance, given by the form $d^2(x, y) = (x - y)^\top M(x - y)$, where M is the precision matrix capturing the variances and covariances of the feature variables. A plethora of methods have been proposed to learn M from pairwise constraints which indicate whether a pair of objects belong to a same class [5, 6, 17, 18]. Among them the Information Theoretic Metric Learning (ITML) [5] is one state-of-the-art method. However, none of these methods serves our purpose of maximally differentiating different users by leveraging multifaceted features. Specifically, (1) for a single facet, Mahalanobis distance $d^2(x, y)$ is a special case of $f(x, y)$ in our design, as $d^2(x, y)$ only captures ellipsoid decision boundaries while $f(x, y)$ captures any second order decision boundaries; (2) $d^2(x, y)$ cannot encode the information from multiple facets. One straightforward solution may be to concatenate all the facets and learn the Mahalanobis distance on the concatenated feature vectors. The performance of this solution however could be arbitrarily bad because of the distinct characteristics of different facets; meanwhile this solution may lead to large storage and time costs as it requires to learn a tremendously large M.

Our footprint matching framework is related to the problem of nearest neighbor search in inner-product space [15, 13]. This work is innovative in the derivation of the equivalence between multifaceted footprint matching and maximum inner-product search.

8 Conclusion

This work tackles the challenging problem of attributing network traffic for security forensic analysis. We present a novel footprinting model that captures user- and context-sensitive patterns from multiple facets of network traffic. Extensive experiments using data from a large-scale enterprise network demonstrate the benefits of multifaceted footprinting including scalable and accurate traffic attribution. This work also opens several directions for further investigation: 1) incorporating domain knowledges in training multifaceted footprints; 2) exploring alternative feature models of network measures, e.g., Markov chain; 3) exploiting more complicated context models, e.g., multi-layer context network; and 4) designing online or incremental version of footprint learning algorithms.

References

[1] M. Afanasyev, T. Kohno, J. Ma, N. Murphy, S. Savage, A. C. Snoeren, and G. M. Voelker. Privacy-preserving network forensics. *Commun. ACM*, 54:78–87, 2011.

[2] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable internet protocol (aip). In *SIGCOMM*, 2008.

[3] F. Bergadano, D. Gunetti, and C. Picardi. User authentication through keystroke dynamics. *ACM Trans. Inf. Syst. Secur.*, 5:367–397, 2002.

[4] J. A. Calandrino, W. Clarkson, and E. W. Felten. Bubble trouble: off-line de-anonymization of bubble forms. In *SEC*, 2011.

[5] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. Information-theoretic metric learning. In *ICML*, 2007.

[6] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighborhood component analysis. In *NIPS*, 2004.

[7] M. Guillaumin, J. Verbeek, and C. Schmid. Is that you? metric learning approaches for face identification. In *ICCV*, 2009.

[8] D. J. Hand and R. J. Till. A simple generalisation of the area under the roc curve for multiple class classification problems. *Mach. Learn.*, 45(2):171–186, 2001.

[9] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft. Structural analysis of network traffic flows. In *SIGMETRICS/Performance*, 2004.

[10] Z. Li, L. Cao, S. Chang, J. R. Smith, and T. S. Huang. Beyond mahalanobis distance: Learning second-order discriminant function for people verification. In *CVPR Workshops*, 2012.

[11] Q. Mei and C. Zhai. A mixture model for contextual text mining. In *KDD*, 2006.

[12] F. Monrose, M. K. Reiter, Q. Li, and S. Wetzel. Cryptographic key generation from voice. In *S&P*, 2001.

[13] F. P. Preparata and M. I. Shamos. *Computational geometry: an introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[14] A. Rakotomamonjy, F. R. Bach, S. Canu, and Y. Grandvalet. SimpleMKL. *Journal of Machine Learning Research*, 9:2491–2521, 2008.

[15] P. Ram and A. G. Gray. Maximum inner-product search using cone trees. In *KDD*, 2012.

[16] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. In *CCS*, 2002.

[17] F. Wang. Semisupervised metric learning by maximizing constraint margin. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 41(4):931–939, 2011.

[18] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems 15*, volume 15, pages 505–512, 2002.

[19] L. Yang and R. Jin. Distance metric learning: A comprehensive survey. Technical report, Michigan State University, 2006.

[20] N. Zheng, A. Paloski, and H. Wang. An efficient user verification system via mouse movements. In *CCS*, 2011.