Computer Networks 129 (2017) 178–192

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Internet traffic classification based on expanding vector of flow☆

Lei Ding[a,*], Jun Liu[a], Tao Qin[a], Haifei Li[b]

[a] MOEKLINNS Lab, Department of Computer Science and Technology, Xi'an Jiaotong University, 710049, China
[b] Department of Computer Science, Union University, Jackson, TN 38305, USA

ARTICLE INFO

Article history:
Received 22 March 2017
Revised 24 September 2017
Accepted 27 September 2017
Available online 28 September 2017

Keywords:
Traffic classification
Flow relationship
Packet loss
Mice flow

ABSTRACT

To reduce the number of packets used in categorizing flows, we propose a new traffic classification method by investigating the relationships between flows instead of considering them individually. Based on the flow identities, we introduce seven types of relationships for a flow and a further Expanding Vector (EV) by searching relevant flows in a particular time window. The proposed Traffic Classification method based on Expanding Vector (TCEV) does not require an inspection of the detailed flow properties, and thus, it can be conducted with a linear complexity of the flow number. The experiments performed on real-world traffic data verify that our method (1) attains as high a performance as the representative methods, while significantly reducing the number of processed packets; (2) is robust against packet loss and the absence of flow direction; and (3) is capable of reaching higher accuracy in the recognition of TCP mice flows.

1. Introduction

1.1. Background

Traffic classification, which associates packets or flows to the generating application, is an essential task in Internet management. It helps in the allocation, control, and monitoring of the usage of resources in networks [1], and plays an important role within a quality-of-service (QoS) enabled network and in the field of network security.

Methods of traffic classification are categorized using three techniques: port-based, payload-based, and behavior-based. In the early stage of Internet development, port numbers used for classifying applications are either assigned by or registered to the Internet Assigned Numbers Authority (IANA) [2]. However, as regards the continuous increase in registered numbers, as well as the use of dynamic ports by many P2P applications and the emergence of tunneling protocol techniques, port-based methods have become less convincing. Moreover, 30%-70% of traffic flows cannot be identified [3].

Payload-based methods, also known as deep packet inspection (DPI), investigate the transferring content in packets and are capable of a high recognition accuracy. However, they also lead to low efficiency, frequent updating of matching rules, and privacy problems [1,4]. Furthermore, an increasing number of applications encrypt their payloads, thereby making the methods unavailable.

At present, methods based on traffic behavior are widely used, and obtain effective performances with various machine learning techniques, such as Naïve Bayes, k-NN, Support Vector Machine (SVM), and Decision Tree [4–6]. In these methods, flow is often represented by features extracted from its packets, like the average packet size and the average inter-arrival time. Although the current behavior-based methods have many advantages, some weaknesses remain, especially for real-time classification. First, the extraction of many features must check the network and transport layer headers of every packet in the flow, a process that involves large computational and memory overheads. Second, certain features like maximal packet size are a posteriori, because they are calculated until the flow terminates. Third, it remains a challenge in mice flow classification because these flows possess too few packets to collect representative features, thereby decreasing the total accuracy of the classification [4].

In some works, researchers use the similarity between flows (in IP addresses and port numbers) to assist the behavior-based methods. For example, flows at the same destination side (i.e., those with the same IP address, port number, and transport layer proto-

☆ The research was supported in part by National Key Research and Development Program of China (2016YFB1000903), National Natural Science Foundation of China (61672419, 61532004, 61532015), Innovative Research Group of the National Natural Science Foundation of China (61721002), Innovation Research Team of Ministry of Education (IRT_17R86), Natural Science Foundation of Shaanxi Province (2016JM6040) and the Project of China Knowledge Centre for Engineering Science and Technology.

* Corresponding author.
E-mail addresses: lding@sei.xjtu.edu.cn (L. Ding), liukeen@mail.xjtu.edu.cn (J. Liu), tqin@sei.xjtu.edu.cn (T. Qin), hli@uu.edu (H. Li).

https://doi.org/10.1016/j.comnet.2017.09.015
1389-1286/

col) are supposed to be generated by the same application [7]. Although this heuristic helps improve the classification performance, the methods are still based on flow features mentioned above and suffer from the shortcomings of typical behavior-based methods.

1.2. Overview of the proposed method

To reduce the number of processed packets required for classification, we introduce the concept of flow relationship. If two flows have a type of relationship, they are considered to share one or some identities (i.e., the source/destination IP addresses and port numbers). Hence, we obtain a group of relationships in a much generalized way, unlike previous works that only use destination side heuristics. For each type of relationship, we build a flow set with the relevant flows; this process is called flow expanding. Finally, we derive a vector for a flow, named expanding vector (EV), by countering the flow number in each set. By using the vector, we propose a new Traffic Classification method based on Expanded Vector (TCEV).

In contrast to the typical behavior-based methods, TCEV inspects neither the size nor the inter-arrival time of each packet in the flow, but only the relationships between flows. In other words, TCEV is a classification method that is entirely based on flows.

1.3. Contributions

Our main contributions are listed below.

1. We introduce seven types of relationships between flows by investigating their IP addresses, port numbers, and starting timestamps. Given the relevant flows, we then define a flow's expanding vector to represent the state upon its establishment.
2. We propose a new traffic classification method based on expanding vector called TCEV. The method does not need to inspect detailed flow properties and can be performed synchronously with the aggregation of flows. Thus, TCEV can impressively reduce the overheads of feature collection. Furthermore, TCEV is robust to packet loss and the absence of flow direction. It can also improve the recognition accuracy of TCP mice flows.
3. We evaluate our method using large-scale, real-world traffic data. Compared with two state-of-the-art methods and two other widely used methods, we achieve both a satisfactory classification performance and the lowest overheads.

The remainder of the paper is organized as follows. In Section 2, we recall the current research on traffic classification and the utilization of destination side heuristics. We provide the notations of flow relationships, flow expanding and expanding vector in Section 3, and analyze the effect of flow expanding on applications in Section 4. The proposed method is shown in Section 5. In Section 6, we evaluate the method on both a binary and a multi classification with comparisons of four baseline methods. Special situations are also discussed. Finally, we present our conclusions and directions for future works in Section 7.

2. Related works

2.1. Behavior-based classification

Behavior-based classification is always combined with machine learning methods [4]. In the supervised scheme, the classifier is trained on a set of a priori labeled flows. Then, an unknown instance is recognized as a particular class of application. Roughan et al. [8] used the nearest neighbor (NN) algorithms to classify seven applications (i.e., DNS, FTP-data, HTTPS, Kazaa, RealMedia, Telnet, and WWW) and achieved an error rate of 9.4% to 12.6%. By using the Naïve Bayes method, Moore and Zuev [9] obtained a relatively high trust value for each class. Later, they improved their work with the Bayesian Neural Network (BNN) and achieved up to 99% accuracy on the dataset collected in one day [10].

To further improve classification accuracy, the ensemble method is also used. Callado et al. [11] discussed the combination of different base classifiers in five types of networks and provided a utilization recommendation for each situation.

The unsupervised scheme aims to categorize network traffic into clusters automatically. However, the natural clusters likely do not match the classes of applications. Either flows of an application are divided into several clusters, or a cluster contains flows of several applications. McGregor et al. [12] first used an expectation maximization (EM) algorithm to cluster traffic. Zander et al. [13] improved the performance to a median accuracy of more than 80% by randomly repeating EM searches to determine the best clusters. Erman et al. [14] obtained highly improved clusters with DBSCAN.

The semi-supervised method or hybrid method can be used to refine the clusters. In the work of Erman et al. [15], a training set of labeled flows combined with unlabeled flows is fed into a K-means clusterer. Then, the cluster is labeled by the training flows within it, finally achieving 94% flow accuracy. Zhang et al. [16] improved a three-stage hybrid method to classify flows of new applications from the unlabeled traffic. The method obtains better performance with these flows not predefined in the training set.

2.2. Overhead reduction

In the above research, when a machine learning method is applied, the flow properties must first be collected, which is always a process of calculating the size and inter-arrival time as well as their statistics and transformations, on all packets in the flow. The overheads of feature collection also become an important issue, especially in real-time situations [17]. Typically, two solutions are applied: feature reduction and packet reduction.

For feature reduction, Li et al. [18] analyzed the time and space complexity of collecting twelve typical features. Fahad et al. [19] discussed the prevalent feature selection techniques on 248 flow features using the criteria of goodness, stability, and similarity. Park et al. [20] proposed a new feature selection method based on genetic algorithm (GA) and achieved an accuracy of 95.3% with the C4.5 classifier.

In packet reduction, one common approach is calculating flow features only by the first few packets. Bernaille et al. [21] used the first four to five packets to classify TCP-based applications. Their results show that more than 80% of total flows are correctly identified, but almost all POP3 flows are misclassified. Although collecting flow features via the first few packets is attractive, the classifier is quite sensitive to the loss or disorder of these packets, which cannot be guaranteed in the real-time network. To improve this, Nguyen et al. [22] proposed the sub-flow, which is a small number of most recent packets taken at any point in a flow's lifetime. They reached up to 99% precision and 95% recall in the classification of an online game and VoIP. However, the approach is still a tradeoff between overhead and performance.

Another approach for packet reduction is sampling, which can dramatically reduce the processed flows and packets. However, many properties are difficult to be estimated from the sampled trace. For example, Murai et al. [23] argued that the flow length distribution is unrecoverable if less than half of the packets are sampled.

2.3. Utilization of destination side heuristics

In the normal behavior-based methods, flows are always treated individually. Properties are extracted from the interior packets of a

flow itself. The heuristics of the similarity between flows, typically the destination side information, are also used in the classification.

Ma et al. [24] first introduced a service key to combine clusters, which is defined as a three-tuple (responder address, responder port, and transport protocol). The service key is assumed to share the same target application, as hosts typically communicate with servers at specified address-port pairs. Canini et al. [7] also considered that packets with the same three-tuple fingerprint belong to the same application.

Zhang et al. [25,26] used the destination side heuristic to improve the classification results. They introduced a notation of bag-of-flows (BoF), which is a set of flows sharing the same three-tuple. Then, they proposed a traffic classification method using correlation information. The method combines the predictions of flows in the same BoF and assigns the result to the whole BoF, and finally, achieves 10%–15% improvement in *F*-measure for certain classes like POP3 and DNS. The equivalence set constraints in [27], the service-based statistics in [28], and the side information in [29] draw similar heuristics as well.

Although the heuristics help refine the *F*-measure or overall accuracy in the classification of target applications, these methods remain based on the statistics of detailed information of the interior packets of flows.

3. Notations

In this section, we propose a broad set of flow relationships aside from considering the destination side. Based on each type of relationship, we introduce an expanding set for a flow *f*. Then, we construct an expanding vector (EV) in which each element can represent the number of flows having a certain relationship with *f* in a particular time window.

3.1. Flow relationships

As a de facto industry standard, Netflow given by Cisco [30] defines a flow as a series of successive packets that share the same five-tuple (src_{ip}, src_{port}, dst_{ip}, dst_{port}, *protocol*) with a specified timeout, where src_{ip} and src_{port} are the identities (IP address and port number) on the source side, dst_{ip} and dst_{port} are the identities on the destination side, and *protocol* is involved with the transport layer protocol. In the rest of the paper, we consider only TCP and UDP flows, and omit the field *protocol*[1] for convenience. Therefore, we use the first four fields (identities) as well as the starting time *t* to represent a flow.

A flow can also be considered as the communication between two network endpoints, which are known as the *service access points* (SAP). Then, a four-tuple-modeled flow (without protocol) can be considered as a series of packets that belong to the pairwise source/destination *transport service access points* (TSAP, which normally refers to an IP address and the using port number) or *network service access points* (NSAP, which normally refers to an IP address).

Given two flows, *f* and *g*, their relationships can be defined by whether they have one or both communicating SAPs. Consequently, we have seven types of relationships shown in Table 1.

In the aliases, the prefix *L* and the middle number give how many identities shared by the flows, and also indicate the *level* in the hierarchy of flow relationship shown in Fig. 1. The suffix *SRC* is involved with the source side, and *DST* is for the destination side. When *f* and *g* have a type of relationship, *g* is said to be a *relevant* flow of *f*. For convenience, we consider that the flow *f* itself, which can be represented as (TSAP, TSAP), satisfies all the seven types of relationships.

Table 1
Types of relationships between flow *f* and *g*.

Alias	Sharing SAPs	Sharing identities
L3SRC	(TSAP, NSAP)	src_{ip}, src_{port}, dst_{ip}
L3DST	(NSAP, TSAP)	src_{ip}, dst_{ip}, dst_{port}
L2+	(NSAP, NSAP)	src_{ip}, dst_{ip}
L2SRC	(TSAP)	src_{ip}, src_{port}
L2DST	(TSAP)	dst_{ip}, dst_{port}
L1SRC	(NSAP)	src_{ip}
L1DST	(NSAP)	dst_{ip}

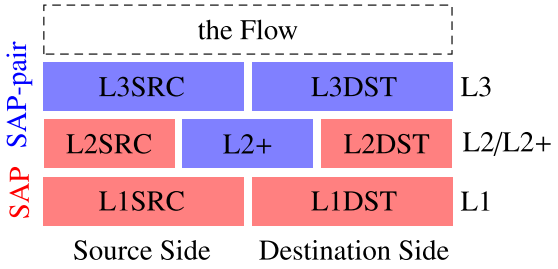


Fig. 1. Hierarchy of flow relationship.

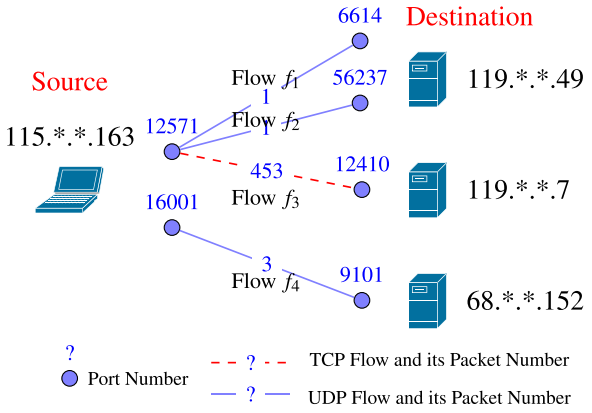


Fig. 2. An example of BitTorrent flows generated by Xunlei in a 3-s period.

The hierarchy of flow relationship includes three levels: L1, L2/L2+, and L3. Considering relationships at the same side (source or destination), the upper level in the hierarchy can be seen as a strong relationship, whereas the lower level is a weak relationship. Naturally, if two flows satisfy a strong relationship, they also satisfy a weak relationship. Moreover, the L1 and L2 relationships can be considered as *SAP-oriented*, whereas the L2+ and L3 relationships are *SAP-pair-oriented*, according to whether both sides or only a single side are involved.

To provide an intuitive illustration, we use a small piece of BitTorrent traffic as an example which is shown in Fig. 2. Four flows are generated in 3 s by a Xunlei desktop client[2] located at *115.*.*.163*, which connect the client (the source) to three servers (the destinations). Flows f_1, f_2 and f_3 use the same port number 12571 at the source side. f_1 and f_2 connect to the same destination, while f_3 connects to another. Flow f_4 uses a different source port 16001. According to the flow relationship definitions, f_1 is L3SRC (also L2+) relevant to f_2; the three flows f_1, f_2 and f_3 are L2SRC relevant; and all the four flows are L1SRC relevant to one another.

[1] Most of the Internet traffic are TCP and UDP flows. According to our trace $\mathcal{T}$ in Section 6.1.1, flows with the same four-tuple but with different TCP or UDP protocols are rare, with a percentage of about 9.75e−5.

[2] http://www.xunlei.com, which is a BitTorrent client prevalent in China.

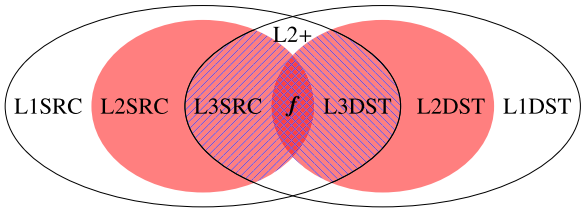


Fig. 3. Venn diagram of flow f's expanding sets, where each set is labeled by the generating relationship.

3.2. Expanding vector of a flow

Given a flow and a type of relationship, we can use the relevant flows to make up a set, which can be regarded as an *expansion* of the flow. Now, we restrict the expansion to relevant neighbor flows. If flow f and g are *neighbor* flows, then

$$|t_f - t_g| \leqslant w, \tag{1}$$

where t_f is the starting time of f, i.e., the timestamp of its first packet. Likewise, t_g is the starting time of g. w is a given time window called the *expanding window*. We use neighbor flows to represent the flows generated close to flow f instead of those co-existent with f, because the former can better reflect flow changes when the concerned f emerges.

Considering flow f and an expanding window w, we obtain an *expanding set* for each type of relationship by enumerating relevant neighbor flows. For example, the L3SRC expanding set of f includes all flows that are L3SRC relevant to f and generated in $[t_f - w, t_f + w]$. We use $l_{3s}, l_{3d}, l_{2+}, l_{2s}, l_{2d}, l_{1s}, l_{1d}$ to represent the respective flow numbers (i.e., cardinality) of the seven expanding sets, according to the sequence of relationships listed in Table 1. Then, we obtain the *expanding vector* (EV) of f, which is written as

$$EV(f) = (l_{3s}, l_{3d}, l_{2+}, l_{2s}, l_{2d}, l_{1s}, l_{1d}). \tag{2}$$

Fig. 3 shows the inclusion relations among the expanding sets of a flow according to the hierarchy of flow relationship. As can be seen, the EV elements have a certain order in terms of size.

As an example, we calculate the expanding vector of flow f_1 in Fig. 2. As f_2 is L3SRC relevant to f_1, the L3SRC expanding set of f_1 has two flows including itself. Thus, we have $l_{3s} = 2$. While no other flows are L3DST relevant to f_1, then $l_{3d} = 1$. Similarly, we calculate other elements and a vector (2,1,2,3,1,4,2) for flow f_1 is finally obtained.

In the rest of the paper, we use the term *expanding* as to find relevant neighbor flows for the flow concerned.

4. Flow expanding

In this section, we discuss the influence of flow expanding from the application view. First, by investigating isolated flows, we show that most flows of the major applications are expandable, that is, we can find at least one relevant flow other than the flow itself. Next, although a flow is expanded to a set of relevant flows, the corresponding applications do not diverge much. These traits make EV a suitable property for classification.

4.1. Isolated flows

Considering a special expanding vector that has an element equivalent to 1, such as $l_{3s} = 1$, this means that the flow has no other L3SRC relevant flows. Correspondingly, such flow is an L3SRC *isolated* flow. The number of isolated flows varies in different applications but impairs the efficiency of expanding when too many isolated flows are found. Next, we check this number after each type of expansion.

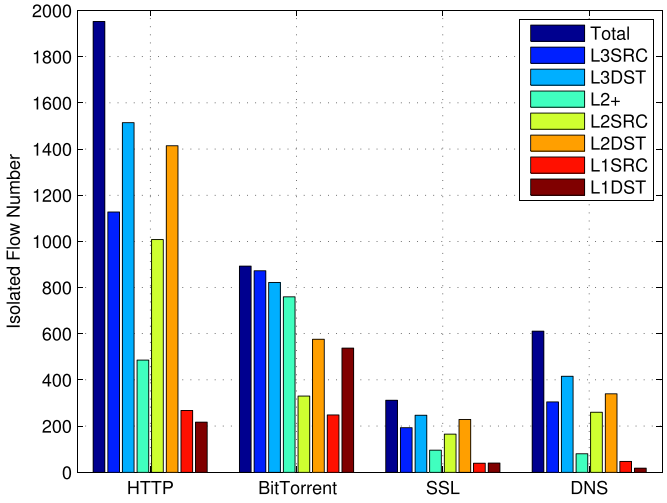


Fig. 4. The variation of isolated flows for four major applications. *Total* refers to the original number of flows of the corresponding application.

We select four major applications in our traffic, namely, HTTP, BitTorrent, SSL, and DNS. Fig. 4 shows the results of the seven types of expansions. We can find that isolated flows decrease with expanding. The number drops notably for the three applications except BitTorrent, especially with L2+ and L1 expansions. For example, the L2+ isolated flows only account for 32% of HTTP, and the L1 isolated flows are much fewer. Furthermore, even for BitTorrent, the number of L1SRC isolated flows is also small.

In fact, for many network applications, a conversation no longer tends to be carried out with only a single flow. Instead, a group of flows is generated to support the communication. On the one hand, the network service is normally supplied by a set of computer clusters. For example, when visiting a website, we would not only connect to the web server but also to the ad server, the image server, and the video server simultaneously. Thus, the client generates many flows that target different IP addresses. On the other hand, even between two hosts, there are also probably many flows either for the acceleration of data transferring or used to send control messages.

The avoidance of isolated flows makes flows expandable, and further feasible to classify flows with expanding.

4.2. Dominant application

Although the generating applications are diversified when a flow is expanded to a set of flows, a dominant application can always be observed in an expanding set. This is evident for an SAP-pair-oriented expanding. When two hosts are connected, an application is always responsible for the communication, and most of the flows between the two hosts belong to the application. Thus, with an SAP-pair-oriented expanding, we would obtain a slightly large flow set, in which one application is still in the majority. In the rest of the subsection, we focus on the situation after an SAP-oriented expanding and discuss the applications on the expanding sets.

4.2.1. Applications in the expanding set of SAP

Before looking into the expanding set of a flow, a natural question is whether there is a dominant application for an SAP. Here, the expanding set of an SAP can be considered as the set of flows sharing the same SAP after the expansion of all flows. In other words, given four kinds of SAPs, namely, $nsap_s = (src^*_{ip})$, $nsap_d =$

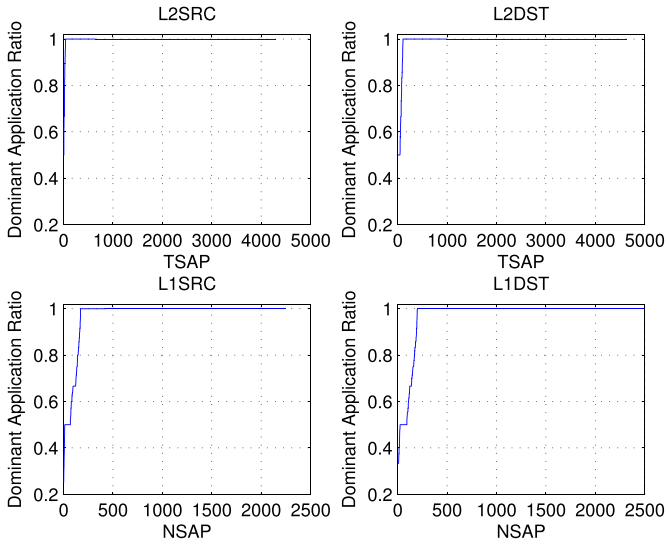


Fig. 5. Distribution of the dominant application in the expanding set of SAPs (SAP-oriented). SAPs are ordered by the dominant application ratio.

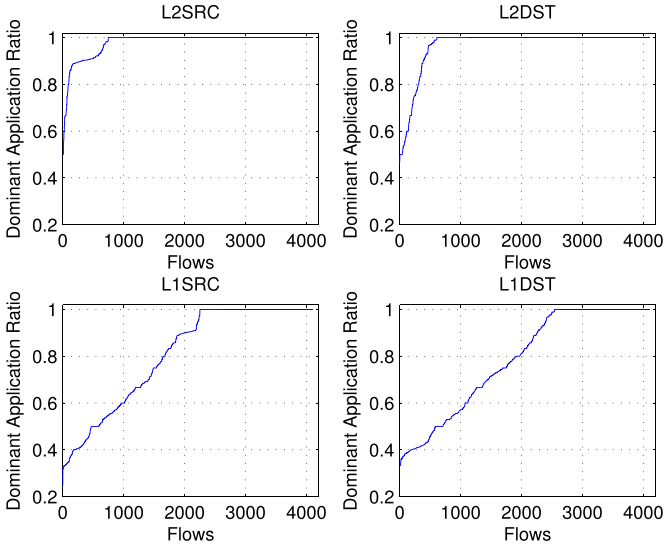


Fig. 6. Distribution of dominant application in the expanding set of flows (SAP-oriented). Flows are ordered by the dominant application ratio.

(dst^*_{ip}) and $tsap_s = (src^*_{ip}, src^*_{port})$, $tsap_d = (dst^*_{ip}, dst^*_{port})$, the expanding sets of an SAP are written as

$$\begin{aligned} ES(nsap_s) &= \{f | f.src_{ip} = src^*_{ip}\}, \\ ES(nsap_d) &= \{f | f.dst_{ip} = dst^*_{ip}\}, \\ ES(tsap_s) &= \{f | f.src_{ip} = src^*_{ip}, f.src_{port} = src^*_{port}\}, \\ ES(tsap_d) &= \{f | f.dst_{ip} = dst^*_{ip}, f.dst_{port} = dst^*_{port}\}, \end{aligned} \tag{3}$$

where f is a flow.

With a 40-s real-time trace randomly selected from our dataset in Section 6.1.1, we collect 4295 TSAPs and 2245 NSAPs at the source, and 4630 TSAPs and 2482 NSAPs at the destination. On each of these SAPs with the corresponding expansion, we calculate the percentage of flows belonging to a dominant application, i.e., the dominant application ratio. The results are shown in Fig. 5. We can find that, in a small period (consistent with the expanding window, here is 15 s), almost all flows on an SAP are dominated by one application. Moreover, most of the SAPs have exactly one application (i.e., 99.1%, 97.7%, 92.4%, and 92.1%, respectively).

According to the results, especially for L2DST expanding shown as the upper-right part of the figure, we can classify a set of flows sharing the same (dst_{ip}, dst_{port}) pair to the same application, which is the basic assumption of previous works [24,25,27–29].

4.2.2. Applications in the expanding set of flow

We examine the applications in the expanding sets of flows. Given a flow f^*, we have

$$\begin{aligned} ES_{1s}(f^*) &= \{f | f.src_{ip} = f^*.src_{ip}\}, \\ ES_{1d}(f^*) &= \{f | f.dst_{ip} = f^*.dst_{ip}\}, \\ ES_{2s}(f^*) &= \{f | f.src_{ip} = f^*.src_{ip}, f.src_{port} = f^*.src_{port}\}, \\ ES_{2d}(f^*) &= \{f | f.dst_{ip} = f^*.dst_{ip}, f.dst_{port} = f^*.dst_{port}\}, \end{aligned} \tag{4}$$

where $ES_{1s}(\cdot)$, $ES_{1d}(\cdot)$, $ES_{2s}(\cdot)$, $ES_{2d}(\cdot)$ are the four expanding sets. The flow numbers in these sets correspond to the last four elements of f^*'s EV.

Excluding those labeled as *Unknown*, we collect 4112 flows from the trace discussed in Section 4.2.1. Fig. 6 shows the distribution of the dominant application. As can be seen, dominant applications still exist for most flows, though the ratios are a little smaller than those of SAPs, particularly after an L1 expansion. Suppose a dominant application contains at least 80% flows in an expanding set, then the dominant ratios are 97.7%, 92.6%, 59.9%, and 53.6% for L2SRC, L2DST, L1SRC, and L1DST expansions, respectively. Furthermore, the *Unknown* flows are not denied but merely excluded.

The aforementioned analysis shows that the application that a flow belongs to is well preserved with expanding. In other words, we can assign a dominant application to the whole expanding set. Thus, the expanding vector, which represents the sizes of the expanding sets of a flow, should be a suitable property for the classification.

5. Traffic classification based on expanding vector

In this section, we introduce the traffic classification method based on expanding vector (TCEV) which is performed by exploring the relationships between a flow and its neighbor flows. We first show the framework of TCEV and a brief illustration of the steps involved. Next, we focus on the EV construction and provide some practical algorithms. Finally, we analyze the complexity of the procedures.

5.1. TCEV framework

Fig. 7 shows the TCEV framework, which consists of three steps, namely, flow aggregation, EV construction, and classification.

First, packets are aggregated into flows according to the five-tuple components with a given timeout. Packet aggregation is a fundamental step in both TCEV and other behavior-based methods. The step is operated on successive packets with two main tasks. First, the five-tuple components are checked to find whether the current packet belongs to an existing flow or starts a new flow. Then, if the five-tuple components match, the arrival timestamp is compared with that of the last packet of the existing flow to determine whether the current packet exceeds the given timeout.

Second, TCEV computes the expanding vector of the flow within an expanding window. This step works on the flow sequence and does not need to look back to the packet sequence. Thereby TCEV can do away with the step of computing packet statistics of each flow, which is essential for most behavior-based methods. Moreover, the step of EV construction can be performed synchronously with flow aggregation because only the flow sequence is used. In the following subsection, we provide the detailed procedures for EV construction.

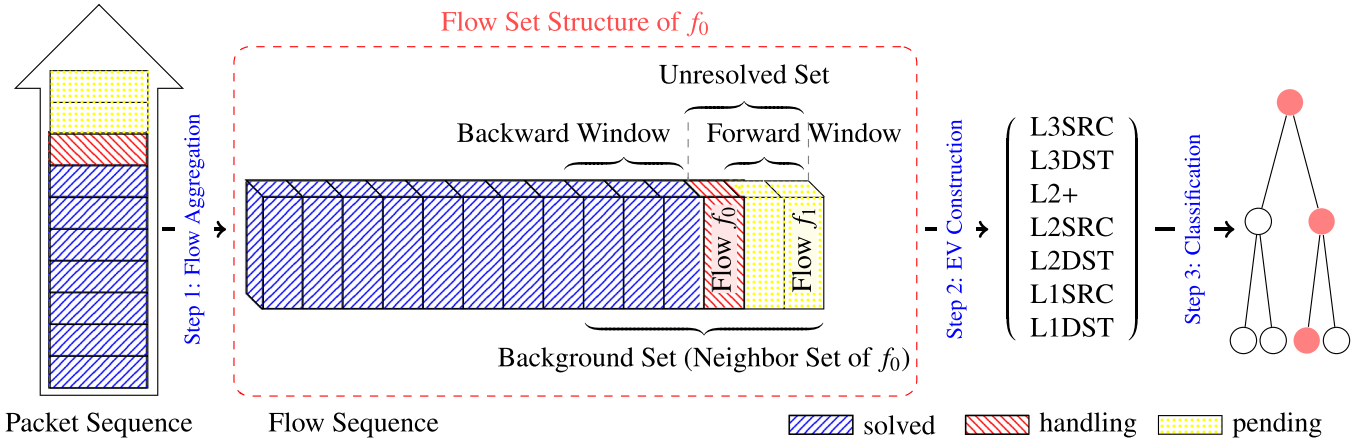


Fig. 7. TCEV framework.

Table 2
Description of function $cond(f_1, f_2, i)$ in Algorithm 1.

i	src_{ip}	src_{port}	dst_{ip}	dst_{port}	Relationship
1	True	True	True	–	L3SRC
2	True	–	True	True	L3DST
3	True	–	True	–	L2+
4	True	True	–	–	L2SRC
5	–	–	True	True	L2DST
6	True	–	–	–	L1SRC
7	–	–	True	–	L1DST

5.2. Expanding vector construction

5.2.1. A straightforward method

Given a flow f, suppose we have collected all its neighbor flows, which compose the *neighbor set* of f (denoted by B). Then, the expanding vector of f can be obtained by checking the identities (IP addresses and port numbers) of all flows in B. This process is shown in Algorithm 1, in which we visit every flow in the neighbor set B to find the relevant flows of f, using function $cond(f_1, f_2, i)$ defined in Table 2 below.

Algorithm 1 Function *flowEV*.

Input: Flow f, Neighbor Set B
Output: $EV(f)$
1: ev is zero filled
2: **for all** $g \in B$, $ev_i \in ev$ **do**
3: **if** $cond(f, g, i)$ is satisfied **then**
4: $ev_i \leftarrow ev_i + 1$;
5: **end if**
6: **end for**
7: $EV(f) = ev$

Function $cond(f_1, f_2, i)$ is used to determine whether the conditions are satisfied for each type of relationship. Here, f_1 and f_2 are two flows, i is presented in the first column. The middle column src_{ip} (also src_{port}, dst_{ip}, dst_{port}) indicates whether f_1 and f_2 have the same src_{ip} (or src_{port}, dst_{ip}, dst_{port}). If the conditions listed in the middle columns are satisfied, then $cond(f_1, f_2, i)$ is true.

The construction of EVs of a trace includes the EVs of all flows in the trace. Here, we take flow f_0 as an example. As shown in Fig. 7, the neighbor set of f_0 is gradually fulfilled as new flow comes. Then, we can calculate the expanding vector of f_0 once it has a complete neighbor set. For the realization, we build two sets. One is the *unresolved set*, denoted by U, where the EVs of flows have not been constructed. The other is called the *background set*, also denoted by B, which contains all the neighbor flows unused yet. Besides, the expanding window is divided into the forward window w_f which represents a period after t_0 (i.e., the starting time of f_0) and a backward window w_b which represents a period before t_0.

Specifically, f_0 is the first flow in U. B stores all the flows since $t_0 - w_b$. As shown in Fig. 7, when f_1 is added, the background set becomes a complete neighbor set of f_0. Then, we calculate the EV of f_0 and remove it from U. Meanwhile, B adjusts the minimal timestamp to fit the backward expanding window of the earliest unresolved flow in U. This process is shown in Algorithm 2, in which flows are successively fetched from an online or offline trace $\mathcal{S}$. The algorithm calls function *flowEV* in Algorithm 1 to output the EV of a flow.

Algorithm 2 EV construction.

Input: $w_f, w_b, \mathcal{S}$
Output: $EV(\mathcal{S})$
1: $U = \emptyset, B = \emptyset$
2: **for all** $g \in \mathcal{S}$ **do**
3: $U = U \cup \{g\}$
4: $B = B \cup \{g\}$
5: **for all** $f \in U$ **do**
6: **if** $f.t + w_f \leq g.t$ **then**// fulfilled neighbor set of f
7: $EV(f) = flowEV(f, B)$// output the EV of f
8: $U = U - \{f\}$
9: $B = B - \{h : h \in B, \nexists h.t < g.t - w_b\}$
10: // remove flows only neighbor to f
11: **end if**
12: **end for**
13: **end for**
14: $EV(\mathcal{S}) = \{EV(f)\}_{f \in \mathcal{S}}$

5.2.2. An improvement

Considering two neighboring flows $f_1, f_2 \in \mathcal{S}$, the comparison of their flow identities would be re-executed in both of their EV constructions. Hence, the straightforward method involves twice as many comparisons as the construction needs.

Besides, EV is additive according to time. Suppose the expanding window of a flow f is divided into two nonoverlapping time slices $w = w_1 + w_2$, then we can calculate EV on each slice and sum up the values with

$$EV_w(f) = EV_{w_1}(f) + EV_{w_2}(f). \quad (5)$$

Consequently, given a time span as the duration of slices, such as 0.5 s, we calculate all the EVs of flows in every slice and then add the EVs of corresponding flows in the slices covered by a

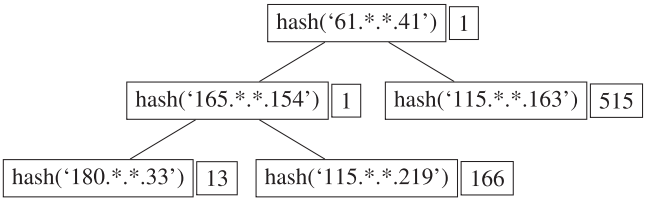


Fig. 8. An example of L1SRC EV tree.

flow's expanding window. We use a binary search tree to record the flow relationships in a slice. Each node of the tree has two fields, one is the hash of a *relationship identity*,[3] which refers to the sharing elements, for example, the (src_{ip}, src_{port}, dst_{ip}) for the L3SRC relationship. The other field is obtained by counting the number of flows sharing the relationship identity in the slice. An example tree for L1SRC is shown in Fig. 8.

We set up two threads, one for construction and the other for searching. The construction thread is used for the construction of EV trees which is performed along with the process of flow aggregation. The searching thread is started with a delay of the forward expanding window, in which the EVs of flows are obtained by summing up the values of corresponding nodes in all slice trees within the expanding window.

5.3. Complexity

In this subsection, we provide a brief complexity analysis on the two EV construction algorithms, shorten as *U–B* (Section 5.2.1) and *EVtree* (Section 5.2.2) respectively. For comparison, we also provide the complexity of feature collection in typical behavior-based methods.

5.3.1. Complexity of U–B

According to Algorithm 2, the overheads of EV construction are determined by scales of the unresolved set U and the background set B. Both the two sets are keeping updated. When a new flow comes, it is added to U and B. When a flow has its EV calculated, it is removed from U, and its neighbor flows are also removed from B. Thus, there is probably no remarkable fluctuation in the sizes of U and B (denoted by N_U and N_B). Fig. 9 shows the changes of N_U and N_B according to the incoming flows. Leaving the initialization stage aside, N_U and N_B tend to be constants that are independent of the flow number.

Then, given the input size n which is the flow number of the trace, the time complexity of method *U–B* is

$$C(Alg_{ub}) = O(\eta_U \eta_B \cdot n) = O(n), \tag{6}$$

where η_U and η_B are the upper bounds of N_U and N_B, respectively, which are determined by the expanding window according to Fig. 9.

5.3.2. Complexity of EVtree

In the algorithm with EV tree, the tree number (denoted by N_{tree}) is limited by the expanding window. Suppose the flow number in a slice is N_{slice}, the number of nodes (i.e., unique relationship identities) in a tree would be less than N_{slice}. Then, the complexity of constructing a flow's EV should be less than $N_{tree}\log N_{slice}$, because the average complexity of either inserting or searching for a binary search tree is logarithmic to the number of nodes.

Fig. 10 shows examples of slices with different durations and starting timestamps. We can find that N_{slice} is also independent of processed flows, and is bounded by a certain constant that is determined by the duration. Therefore, *EVtree* also has a complexity of $O(n)$. As there is always $N_{tree}\log N_{slice} < N_U N_B$, *EVtree* is more efficient than *U–B*.

Considering a normal behavior-based method based on only a single property, for example, the average packet size. If a flow has k packets, then it takes k steps ($k-1$ plus operations and 1 division operation) to obtain the property. Thus, the collection of all features is a process with an order of $O(m)$, where m is the packet number of the trace.

[3] When calculating the hash value, an identity is treated as a string by concatenating all the elements.

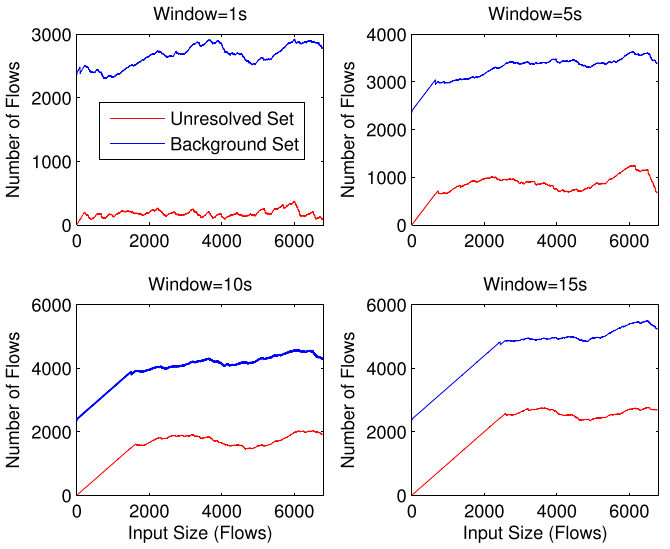


Fig. 9. Sizes of unresolved set U and background set B in Algorithm 2 with different expanding windows. The part of straight line ($y = x$) is an initialization of the set.

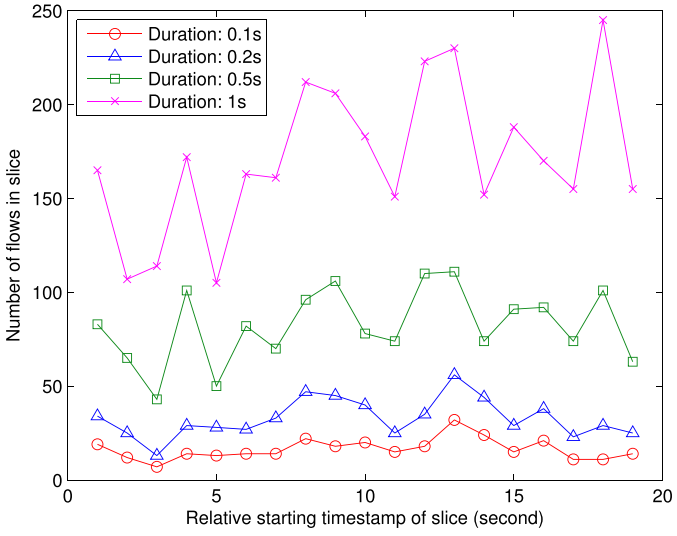


Fig. 10. Number of flows in slices with different durations and starting timestamps. The horizontal axis is the relative timestamp against the beginning of the trace.

6. Evaluation

In this section, we evaluate our method using a real-world captured traffic data. After the description of our dataset and metrics, we conduct the evaluation based on the following questions:

Q1 What is the classification performance of TCEV compared with other methods?
Q2 How can TCEV reduce classification overheads?

Table 3
Description of the three traces.

	trace $\mathcal{A}$	trace $\mathcal{B}$	trace $\mathcal{T}$
.pcap files	1(0.5GB)	1(0.5GB)	89(44.5GB)
Captured	2016	2015	2016
Elapsed	40 s	40 s	1 h 2 s
Flows (IPv4)	13,155	10,100	885,795
Packets	561,247	424,553	51,569,024
Avg. packet/s	13,846.8	10,536.6	14,316.1
Avg. packet size	857	1144	829
Bytes	481M	485M	42,764M
Avg. byte/s	11.8M	12.1M	11.9M
Avg. bit/s	94.9M	96.4M	95.0M

Q3 Does TCEV work without knowledge of flow direction?
Q4 Is TCEV resilient to a terrible network environment like packet loss?
Q5 Can TCEV classify mice flows well?

6.1. Dataset and metrics

6.1.1. Dataset

The real-world traffic data used in the current study is continuously captured at a key egress router located at the northwest regional center of China Education and Research Net (CERNET), with a bandwidth of 10 Gbit/s. Specifically, We collected two large-scale traces. One was captured in November 2016 with a time span of three days. The other was captured in November 2015 with a time span of five days. The traces were preserved in a list of *.pcap* files each of which contained a piece of 40-s traffic. The dataset in the following experiments consists of three parts, which are randomly selected from the captured traces. First, two 40-s subtraces (trace $\mathcal{A}$ and trace $\mathcal{B}$) are used for the binary classification of HTTP and BitTorrent. To better cover the evolution of traffic nature, these two subtraces are selected in different years. In multi-class problems, we use a 1-hour subtrace (trace $\mathcal{T}$), which contains trace $\mathcal{A}$, to obtain more instances. The basic information of the dataset is shown in Table 3.

To obtain the ground truth, flows are carefully labeled by nDPI and manual inspection. nDPI is an open source LGPLv3 library for deep-packet inspection, which is based on *OpenDPI* and includes *ntop* extensions. The version we use is nDPI 1.7,[4] which supports more than 185 protocols. nDPI can also handle some encrypted content with a decoder for SSL certificates.

In our experiments, only IPv4 flows are considered. Fig. 11 shows the distribution of applications in Trace $\mathcal{A}$. The classes are quite imbalanced according to the figure. Among the recognized flows, HTTP accounts for 40.6%, and the top five applications, HTTP, BitTorrent, DNS, ICMP, and SSL, account for 87.5%.

6.1.2. Preprocessing

The preprocessing involves three steps to collect instances from the raw captured traffic data. In this subsection, we would take the preprocessing of trace $\mathcal{A}$ for example.

First, as each captured *.pcap* file only lasts 40 s, a flow aggregated from a file is also limited in 40 s, thus would be a segment of the original flow. We perform nDPI on these small traces and then combine both flow segments and the nDPI results. The combination of flow segments is according to the five-tuple, while the combination of nDPI results is by voting.

Second, we remove the repetitive flows which have already existed before trace $\mathcal{A}$ because only the new flows contribute to the expanding, as shown in Section 5.2. Besides, flows without a TCP or UDP transport protocol, as well as some network layer protocols like ICMP,[5] are also excluded. We denote the refined trace by a bold symbol **A** for trace $\mathcal{A}$. Table 4 shows a brief comparison of trace $\mathcal{A}$ and the refined trace **A**.

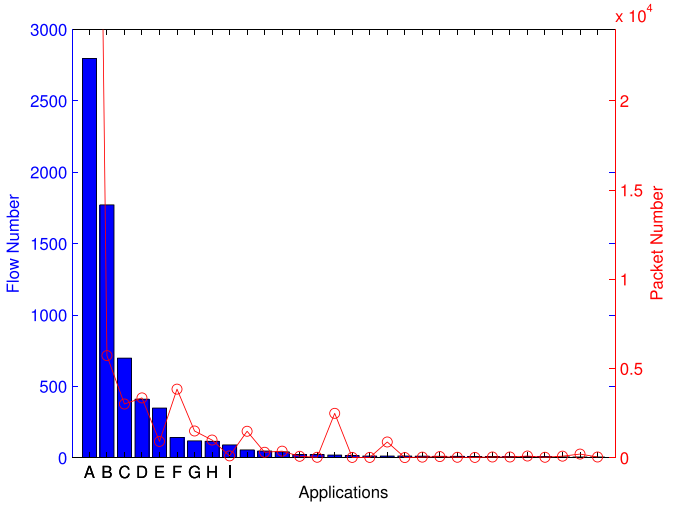


Fig. 11. Application distribution in trace $\mathcal{A}$. The marked applications A to I, in sequence, are HTTP, BitTorrent, DNS, SSL, ICMP, Apple, HTTP Proxy, Quic, and DCE-RPC.

Table 4
Comparison of trace $\mathcal{A}$ and its refined trace **A** in several major applications.

Flows	trace $\mathcal{A}$	trace **A**
HTTP	2796	2254
BitTorrent	1771	1624
DNS	698	650
SSL	411	287
ICMP	348	0
Unknown	5950	4430
Total	12,836	10,168

Third, in the experiments of Section 6.2, the forward and backward expanding time windows are set to be less than 40 s for efficiency. Thus, we need to check the two 40-s neighbor traces before and after trace $\mathcal{A}$, denote as $\mathcal{A}_-$ and $\mathcal{A}_+$, respectively. The two neighbor traces are also refined as trace $\mathbf{A}_-$ and $\mathbf{A}_+$. Finally, we obtain a 120-s, refined trace $\tilde{\mathbf{A}}$ for feature extraction.

6.1.3. Metrics

We use Roman numerals *I, II,* $\cdots$ to represent the applications (i.e., the classes). Table 5 presents the metrics used to evaluate the classification performance of a given class *I*.

According to Section 5.3, we introduce the *number of processed packets* Γ to evaluate the overheads of collecting features. Given a trace $\mathcal{S}$ (i.e., a set of flows), $\Gamma(\mathcal{S})$ provides the number of packets that are used during the feature collection of all the flows in $\mathcal{S}$. Accordingly, we have

$$\Gamma(\mathcal{S}) \leqslant \sum_{f \in \mathcal{S}} \Gamma(f), \tag{7}$$

where f is a flow. The equality always holds for most of the methods based on packet statistics. However, we can derive the inequality form with TCEV, because the neighbor sets are usually overlapped for successive flows in the trace.

[4] http://www.ntop.org/products/deep-packet-inspection/ndpi/.

[5] As a network layer protocol, ICMP is not typically used to exchange data between systems, nor is it regularly employed by end-user network applications [31]. An ICMP flow cannot even be represented with the five-tuple model. Thus, it is often excluded by existing classification methods [1].

Table 5
Metrics for the classification of *I*.

Term	Abbr	Description
True Positive	TP	Flows of *I* correctly classified
True Negative	TN	Flows of not *I* correctly classified
False Negative	FN	Flows of *I* wrongly classified
False Positive	FP	Flow wrongly classified as *I*
True Positive Rate	TPR	TP/(TP+FN)
False Positive Rate	FPR	FP/(FP+TN)
Precision	p	TP/(TP+FP)
Recall	r	TP/(TP+FN)
F-measure	F_1	2rp/(r+p)
ROC Area	AUC	Area under ROC curve[a]

[a] See [32] for detailed description and computing formula of AUC.

Table 6
Average metrics of the five classifiers with the best highlighted in bold.

Classifier	TPR	FPR	p	r	F1	AUC
J48	0.986	0.016	0.986	0.986	0.986	0.993
Random Forest	**0.989**	**0.012**	**0.989**	**0.989**	**0.989**	**0.998**
IB1	0.984	0.018	0.984	0.984	0.984	0.983
Naïve Bayes	0.793	0.154	0.852	0.793	0.792	0.974
Logistic	0.959	0.046	0.959	0.959	0.959	0.988

Considering the strategy of packet reduction, we expect Γ to be a fixed multiple of the flow number. Then, with the linear regression

$$\Gamma(\mathcal{S}) = \gamma|\mathcal{S}| + b, \tag{8}$$

where $|\mathcal{S}|$ represents the flow number, we also use the coefficient γ to describe the overheads.

6.2. Classification of HTTP and BitTorrent

In this subsection, we evaluate our method on a binary classification of HTTP and BitTorrent, also known as the two most popular applications in our traffic.

6.2.1. Performances of different classifiers

For the initialization, we set $w_f = w_b = 40$ (s), same as the duration of trace $\mathcal{A}$. Then all the EVs are constructed according to the longer trace $\tilde{\mathbf{A}}$. Five classifiers are used with 10-folder cross-validation. Table 6 shows the metrics. From the results, TCEV can classify HTTP and BitTorrent flows well. The accuracy and recall rates are both more than 98% for C4.5, Random Forest, and 1-NN classifiers. Among these classifiers, Random Forest [33], which is the ensemble of a multitude of decision trees based on feature bagging, achieves the best performance. In comparison, Naïve Bayes performs poorly because the EV elements are not independent as shown in the Venn diagram in Fig. 3.

6.2.2. Variation of expanding window

The expansion of a flow is performed within a given time window, which is initialized as 40 s for both the forward and the backward. We first shift the value while supposing $w_f = w_b$. Besides, we also set $w_b > 1$, as a too small window makes that no other relevant flow can be found. Fig. 12 shows the classification results with Random Forest.

For $w_b \geq 5$, the classifier can achieve an *F*-measure of larger than 96%, and the performance changes little when $w_b > 25$. According to Section 5.3.2, the expanding window is proportional to the number of slice trees. Thus, a large expanding window would require more computational and memory overheads. From Fig. 12, a backward (also forward) expanding window of 25 s is suitable for both classification performance and overheads.

Because of the forward expanding window, the recognition of a flow suffers a delay in checking the subsequent flows that have not yet generated. To improve the ability of timely classification, we reduce the forward window while keeping a fixed backward window. Accordingly, we set the backward expanding window to 25 s and tune the forward window from 25 s to 0. Fig. 13 shows the performances with the Random Forest classifier. As can be seen, the classifier achieves an *F*-measure of more than 97% even without a forward expanding window (to be 0). In other words, we can identify HTTP and BitTorrent immediately on the first packet of a new flow by only checking the flows already generated during a small period.

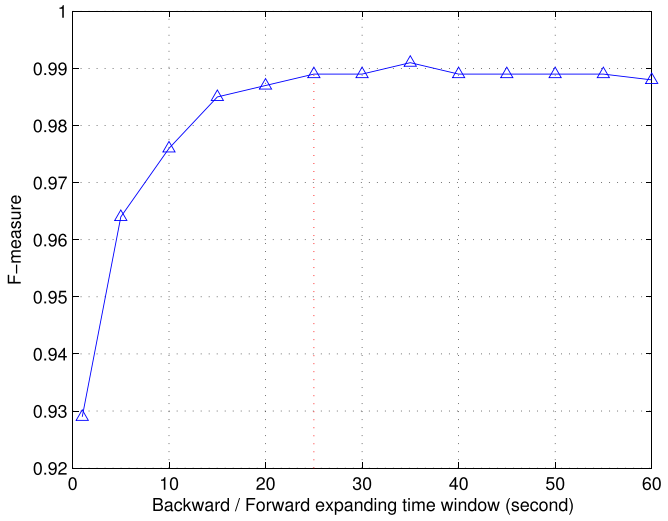


Fig. 12. Performances with different expanding time windows.

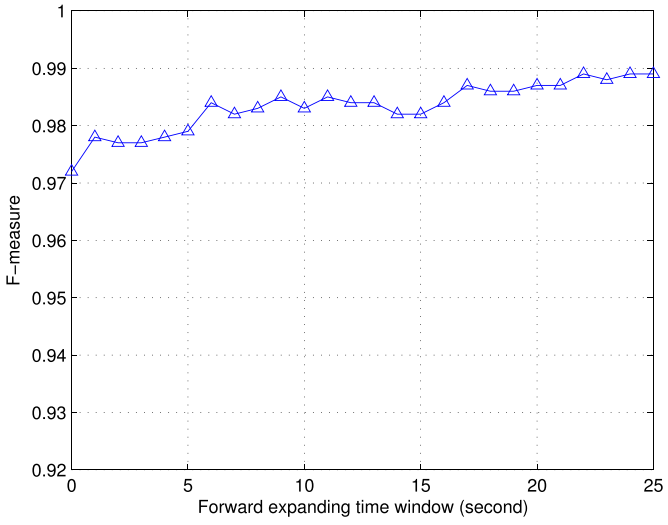


Fig. 13. Performances with a backward expanding window of 25 s and different forward expanding windows.

6.2.3. Comparison with other methods

In this subsection, we conduct comparisons between TCEV and four representative methods. Here, TCEV is performed with the Random Forest classifier and $w_f = w_b = 25$ according to Sections 6.2.1 and 6.2.2.

We first introduce two state-of-the-art methods. One method uses the destination side heuristic mentioned in Section 2.3, named as Bag-of-Flow based Naïve Bayes classification (BoF-NB) [25]. The other method is a packet-reduction method called Synthetic Sub-flow Pairs with Assistance of Clustering Techniques (SSP-ACT) [22] which is mentioned in Section 2.2.

Table 7
Results of TCEV (in bold) and the comparing methods. *sts* represents supplied testing set, and *cv* for cross-validation.

Feature	TPR	FPR	p	r	F1	AUC
BoF-NB(sts)	0.949	0.051	0.957	0.949	0.953	0.978
SSP-ACT(sts)	0.941	0.093	0.941	0.941	0.941	0.950
FIRST(sts)	0.951	0.068	0.951	0.951	0.951	0.972
FULL(sts)	0.943	0.074	0.944	0.943	0.944	0.963
TCEV(sts)	**0.955**	**0.046**	**0.957**	**0.955**	**0.956**	**0.985**
BoF-NB(cv)	0.977	0.023	0.980	0.977	0.978	0.994
SSP-ACT(cv)	0.970	0.045	0.970	0.970	0.970	0.980
FIRST(cv)	0.974	0.028	0.974	0.974	0.974	0.986
FULL(cv)	0.967	0.041	0.967	0.967	0.967	0.83
TCEV(cv)	**0.981**	**0.024**	**0.981**	**0.981**	**0.981**	**0.994**

In BoF-NB, flows are aggregated into different bags according to the destination information. This heuristic assists the combination of classification results. Finally, the flow bags are assigned to different applications. In SSP-ACT, the packets of a flow are divided into various slices which are actually classified. The slices are first clustered to pick up the representatives and then fed into a classifier.

We also introduce another two widely used methods. In the FIRST method, we use the first five packets of a flow to extract the attributes, as suggested by Bernaille et al.citebernaille2006traffic. In the FULL method, we use all packets for the classification like most of the previous works [4].

We use ten features (packet/byte number, the maximum, minimum, mean and standard deviation of packet size/inter-arrival time) for BoF-NB [25] and FULL, and twelve features (the maximum, minimum, mean and standard deviation of packet size/inter-packet size/inter-arrival time) for SSP-ACT [22], and the packet size series for FIRST [21]. The slice divisions in SSP-ACT are also the same as [22], and with clustering, we select six sub-flow positions for HTTP and three for BitTorrent. Table 7 shows the results which contain two groups of experiments. In the first, we use flows in trace **A** for training and trace **B** for testing (denoted by *sts*). In the second, we mix the flows from both trace **A** and **B** and classify with 10-folder cross-validation (denoted by *cv*).

From the table, TCEV achieves reliable performances as the other methods in both sets of experiments, where the *F*-measure values are 0.956 and 0.981 respectively.

Although all the methods are based on flows, TCEV exhibits a different feature collection from the other four. Specifically, TCEV needs only one packet for each flow, while the others investigate more to calculate the statistics. To illustrate this characteristic, we use a series of flow sets to simulate the handling of trace $\mathcal{A}$ by successively adding flows into the former set. Fig. 14 shows the cumulative numbers of processed packets (Γ) for the five methods. Here, the SSP-ACT method uses slices derived from the clusters of both HTTP and BitTorrent.

For FULL and BoF-NB, the numbers increase dramatically, as all the packets are utilized. For the two packet-reduction methods SSP-ACT and FIRST, both the selection of slices in the former and the number of first packets in the latter are tradeoffs between the consumption of resources and classification accuracy. According to Eq. (8), we have $\gamma = 4.992$ and $\gamma = 2.290$ for SSP-ACT and FIRST, respectively. While TCEV has $\gamma = 0.949$,[6] indicating a much slower growth of Γ. TCEV asks for more neighbor flows than other methods at the beginning of the classification. However, the processed number of packets gradually maintains its pace with the number of flows, thus making TCEV a much efficient method in the reduction of classification overheads.

[6] Here, we obtain $\gamma < 1$ and a large $b = 7987$ due to the beginning stage of TCEV.

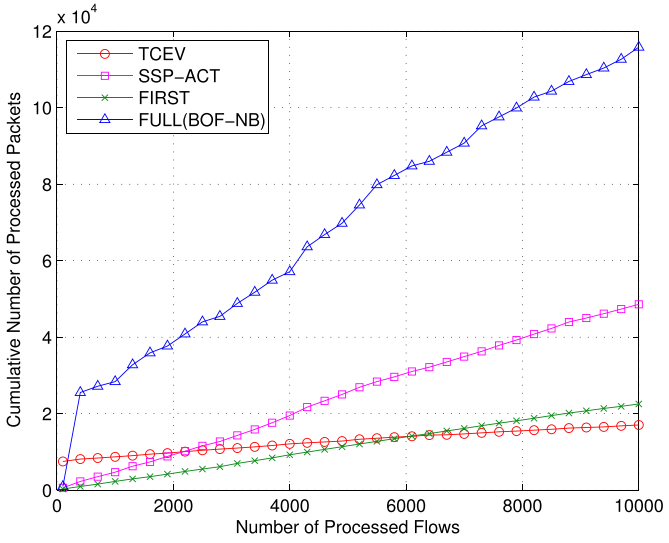


Fig. 14. Cumulative numbers of processed packets (Γ) when handling flows in trace $\mathcal{A}$.

Table 8
Flow numbers of the top six applications.

Application	Flow number
HTTP	233,054
BitTorrent	130,004
DNS	77,200
SSL	30,250
Apple	10,129
HTTP Proxy	10,113

6.3. TCEV on multi applications

In this subsection, we evaluate TCEV on the classification of six major applications in the captured traces. Using nDPI, we label 871,340 flows from trace $\mathcal{T}$ (including *Unknown*), where the top six applications are listed in Table 8. The trace also exhibits a heavy-tailed and highly imbalanced class distribution. The applications that are listed possess 88.7% of the recognized flows, and the largest application HTTP has 23 times as many flows as the smallest application HTTP Proxy.

Due to the imbalance problem, the classifiers tend to be overwhelmed by the majority class and ignore the minority class [34]. To avoid this, we randomly select 2000 flows from each application. Finally, we obtain 20 balanced datasets with the volume of 12,000 instances in every set by repeating this process.

Likewise, TCEV uses the Random Forest classifier with a configuration of $w_f = w_b = 25$ as shown in Sections 6.2.1 and 6.2.2. We also use BoF-NB [25], SSP-ACT [22], FIRST [21], and FULL for comparison.

Fig. 15 shows the classification performances of the six major applications. As can be seen, TCEV also achieves satisfactory performances on the recognitions of the major applications. Specifically, TCEV outperforms FIRST, FULL and SSP-ACT for all the top six applications, and is only a slightly poorer than BOF-NB in recognizing flows of DNS and HTTP Proxy.

Furthermore, the EV property is remarkably complementary to the traditional properties of packet statistics. When enhanced by adding some packet-level information, for example, lengths of the first few packets which are used in FIRST, TCEV can achieve greatly improved performances, shown as TCEV-FIRST in Fig. 15. Especially, the classification performances of DNS and HTTP Proxy are pro-

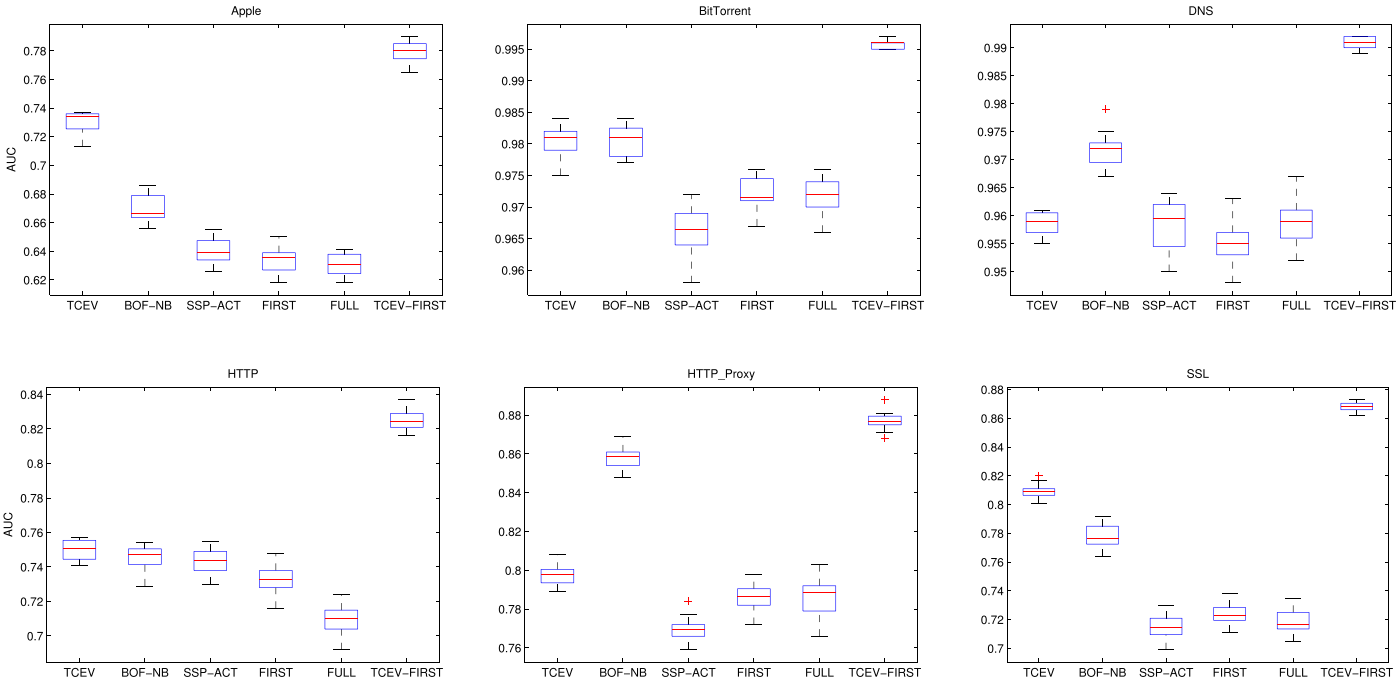


Fig. 15. Classification performances of six major applications with the discussing methods. Each method is performed repeatedly on twenty randomly selected, balanced datasets.

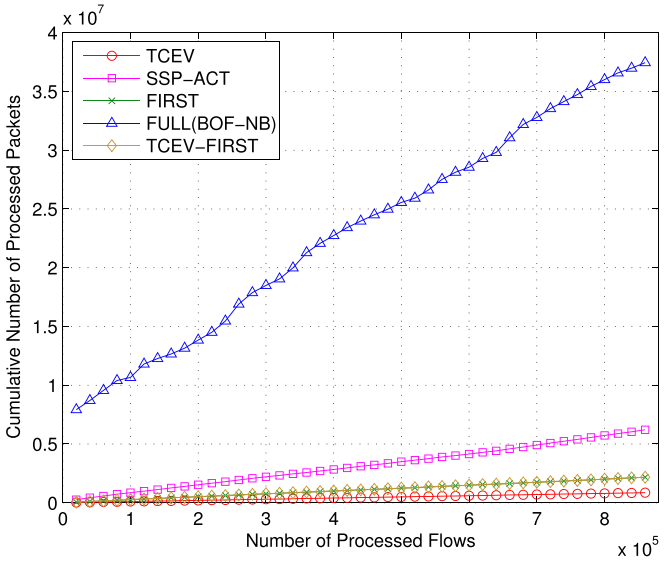


Fig. 16. Cumulative number of processed packets (Γ) when handling flows in trace $\mathcal{T}$.

moted by 3% and 8%, respectively, and are much better than BoF-NB.

Fig. 16 shows the cumulative numbers of processed packets (Γ). We can find that TCEV, with $\gamma = 0.996$, also has the lowest growth rate among the discussing methods. On the contrary, the γ values of FIRST and SSP-ACT are 2.481 and 6.867, respectively. For TCEV-FIRST, the cumulative values of Γ are slightly higher than FIRST. However, the differences gradually decrease as the handling flow sets are enlarged.

Answer to Q1 and Q2: According to Sections 6.2 and 6.3, TCEV significantly reduces the number of processed packets to the number of flows being classified. Moreover, the proposed method also achieves a satisfactory performance compared with the other four representative methods.

6.4. Resilience to flow direction

The direction of a flow is always considered as the direction of its first packet. The source and destination of the flow are also designated as the source and destination of the packet. Flow direction is an important attribute and often plays a key role in many behavior-based methods, such as methods with the destination side heuristic. Besides, the direction of each packet is also used in FIRST [21].

However, checking the direction would bring some risks. The first packet cannot always be captured because of packet loss. The second or another packet may be used to assign the flow direction. Take trace $\tilde{\mathbf{A}}$ (seen in Section 6.1.2) for example, Fig. 17 shows the rate of flows whose second to tenth packet has an opposite direction against the flow. If the first packet is lost (the second is captured), then more than 47% of the flows would be assigned to a wrong direction.

TCEV is a method neutral to the flow direction. What TCEV cares about is whether a relevant flow emerges, regardless of by which packet the flow is discovered, as long as the packet is still in an expanding window.

Hence, we introduce the undirected flow relationships. First, SAPs are alphabetized by their IP addresses. The small IP address (and its port number) is marked as the lower SAP, denoted by *lip* (and *lport*), whereas the big IP address (and its port number) is marked as the upper SAP, denoted by *uip* (and *uport*). Then, given an undirected flow $f = \{lip, lport, uip, uport\}$ and another undirected flow g, the seven types of undirected relationships are defined as follows:

1. **L3LOWER**: as f and g have the same (*lip, lport, uip*).
2. **L3UPPER**: as f and g have the same (*lip, uip, uport*).
3. **L2-**: as f and g have the same (*lip, uip*).
4. **L2LOWER**: as f and g have the same (*lip, lport*).
5. **L2UPPER**: as f and g have the same (*uip, uport*).

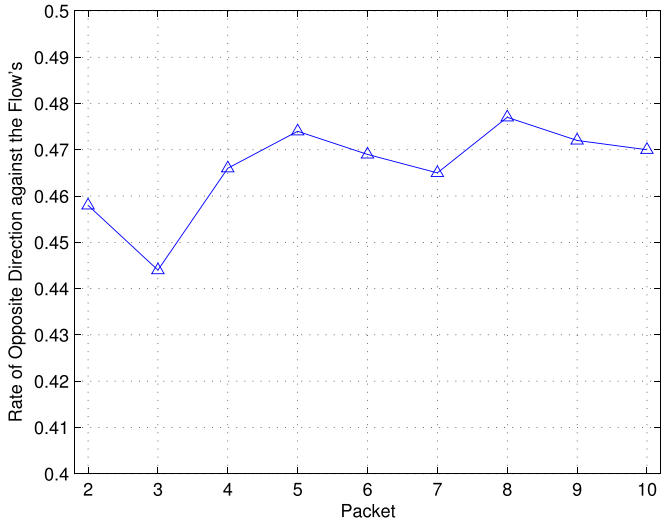


Fig. 17. Percentage of flows whose second to tenth packet has an opposite direction.

Table 9
Results of TCEV with directed and undirected expanding. *sts* represents supplied testing set, and *cv* for cross-validation.

	TPR	FPR	p	r	F1	AUC
directed(cv)	0.981	0.024	0.981	0.981	0.981	0.994
undirected(cv)	0.976	0.030	0.976	0.976	0.976	0.991
directed(sts)	0.955	0.046	0.957	0.955	0.956	0.985
undirected(sts)	0.959	0.063	0.958	0.959	0.958	0.978

6. **L1LOWER**: as f and g have the same *lip*.
7. **L1UPPER**: as f and g have the same *uip*.

Given an expanding window, we use $l_{3l}, l_{3u}, l_{2-}, l_{2l}, l_{2u}, l_{1l}$, and l_{1u} to represent flow numbers of the expanding sets, then the undirected EV of flow f is written as

$$EV_u(f) = (l_{3l}, l_{3u}, l_{2-}, l_{2l}, l_{2u}, l_{1l}, l_{1u}). \quad (9)$$

We artificially create a "mirror" instance f' to eliminate the influence on direction, whose EV is

$$EV_u(f') = (l_{3u}, l_{3l}, l_{2-}, l_{2u}, l_{2l}, l_{1u}, l_{1l}). \quad (10)$$

Table 9 shows the comparison of classifiers with directed and undirected expansions in the classification of BitTorrent and HTTP. The experiments also consist of *cv* and *sts*, and are with 25-s forward and backward windows. The results show that classifications with undirected expandings also achieve as high performances as with directed expandings, and even higher in *sts* experiments.

Answer to Q3: Flow direction is not essential in TCEV. The undirected TCEV works without the direction, but can also achieve high performance.

6.5. Robustness against packet loss

TCEV, FIRST [21], and SSP-ACT [22] are methods aiming at packet reduction. Although they all exhibit an approximately linear complexity of the flow number, the selected packet position plays an important role in the two latter methods. Packet disordering, loss or retransmitting would quite influence the classification performances of SSP-ACT and FIRST. In this subsection, we compare TCEV with FIRST, which is considered to be the most efficient and straightforward approach of packet reduction, under the circumstance of packet loss.

For the simulation, we introduce a random loss rate p, which indicates that each packet in the trace would not be captured with probability p. Considering the classification of BitTorrent and HTTP, we perform packet loss on trace $\tilde{\mathcal{B}}$ (i.e., trace $\mathcal{B}_-$, $\mathcal{B}$ and $\mathcal{B}_+$) with $p = 0.1, \cdots, 0.9$. According to Section 6.1.2, we obtain the refined trace $\mathbf{B}'$ from the flows retained. Then, we predict the applications of flows in trace $\mathbf{B}'$ using the classifier trained on trace $\mathbf{A}$. As a comparison, FIRST is also conducted in the same way.

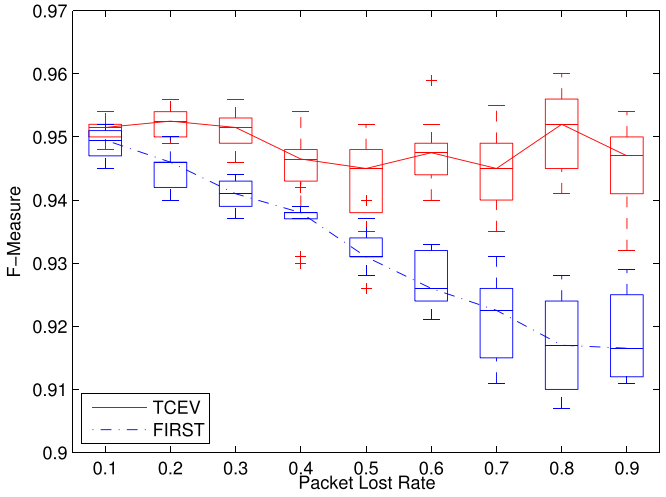


Fig. 18. F-measures of predicting trace $\mathbf{B}'$ under packet loss.

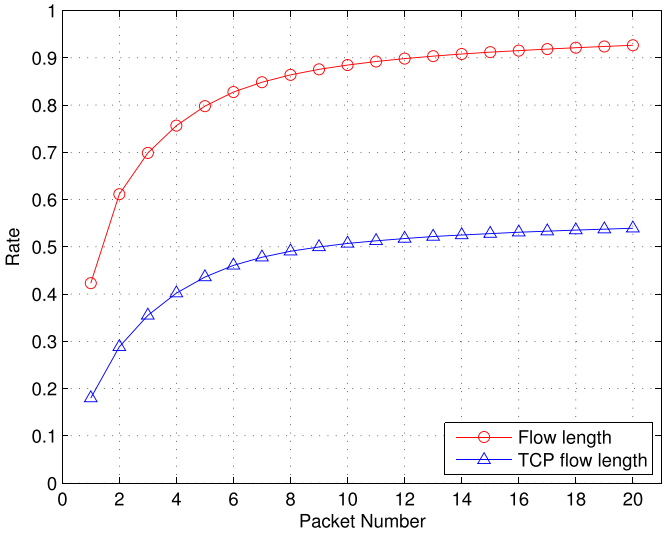


Fig. 19. Flow length distribution of trace $\mathcal{T}$.

The experiments with each packet loss rate are repeated ten times. The classification results are shown in Fig. 18. As can be seen, TCEV works well with a stable F-measure of 0.95 for all the situations. On the contrary, FIRST becomes worse as the loss rate increases, because the positions of the selected packets in a flow in the testing set would probably not match those in the training set.

Answer to Q4: TCEV is robust against packet loss, even if the collected traffic suffers a high loss rate.

6.6. Recognition of TCP mice flows

Mice flows always take a large proportion of network traffic [35]. Fig. 19 shows the flow length[7] distribution of trace $\mathcal{T}$, in

[7] A flow's length is known as the number of packets it contains.

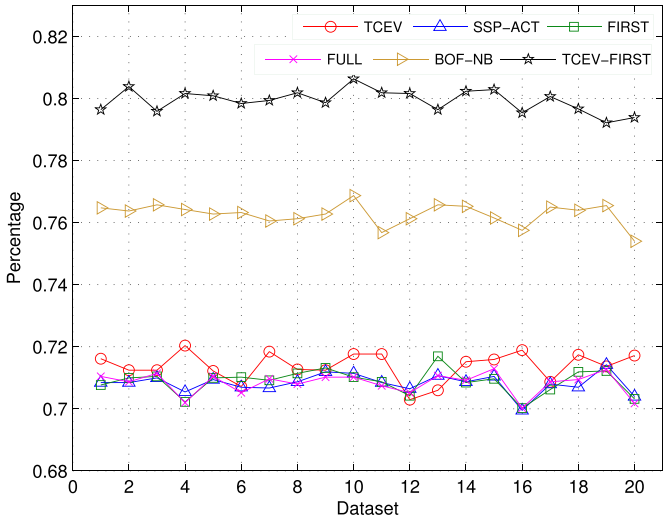


Fig. 20. Percentage of correctly recognized TCP mice flows for different methods.

which approximately 92.6% flows have less than 20 packets, and 1-packet flows even account for more than 40%.

Although the total packets and bytes of mice flows are far less than the elephant flows, it still takes large overheads to discover these flows in the aggregation stage, as explained in Section 5.1. Moreover, the mice flows are usually omitted because the possession of too few packets results in more frequent classification errors [35,36]. In particular, TCP mice flows with less than three packets, which comprise 35.5% of flows in Fig. 19, are normally considered as failed connections and thus excluded from the classification. However, the recognition of these flows could be a valuable reference for understanding the causes of mice flow, and further being beneficial to some network management tasks like intrusion detection.

In TCEV, mice flows work well in the discovery of flow relationships, and in turn, can be recognized with the flow relationships. Here, we use the twenty datasets in Section 6.3 as the training sets, and randomly select another dataset as the test set from a 1-hour-long trace on a different day with an equivalent volume. We compare TCEV with the methods above, and the percentages of correct predictions for TCP mice flows with less than three packets are shown in Fig. 20.

From the figure, TCEV recognizes more TCP mice flows correctly than the methods totally based on packet statistics, e.g., SSP-ACT, FIRST, and FULL. For BoF-NB, the method achieves a better result, because its mechanism can be considered as to promote the performance of mice flows via the results of elephant flows. However, the two-step framework of BoF-NB results in larger computational and memory overheads. If TCEV is combined with some packet statistics, for example, TCEV-FIRST introduced in Section 6.3. Then it can surpass the performance of BoF-NB by approximately 4%, shown as the black line in the figure.

Answer to Q5: TCEV correctly classifies more TCP mice flows than the packet-statistics-based methods. Moreover, the percentage can be improved with very few additional packets.

7. Conclusion

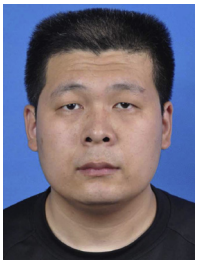
In this work, we construct seven types of flow relationships according to the five-tuple flow model. Based on these relationships, we introduce an expanding vector of flow, which can serve as properties in the proposed classification method TCEV. With real-world traces, TCEV can significantly lessen the number of processed packets to the number of flows being classified, but achieves as high performance as four representative methods in the classification of six major applications, and even performs better in the classification of HTTP and BitTorrent. The complexity of feature collection is reduced to a linear order of the processed flows, which is much smaller than the typical behavior-based methods. TCEV is insensitive to the flow direction and furthermore robust to packet loss. TCEV also performs well in the classification of TCP mice flows which are usually omitted by current methods.

For our future works, we first would like to enhance the concepts of flow relationships by considering more flow characteristics, such as the communicating topology. Next, we would look into the influence brought by the imbalanced application distributions and provide a more reliable strategy for classifying real-world traffic with flow relationships.

References

[1] A. Callado, C. Kamienski, G. Szabó, B.P. Gero, J. Kelner, S. Fernandes, D. Sadok, A survey on internet traffic identification, IEEE Commun. Surv. Tut. 11 (3) (2009) 37–52, doi:10.1109/SURV.2009.090304.
[2] M. Cotton, L. Eggert, J. Touch, M. Westerlund, S. Cheshire, RFC 6335: Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry, Technical Report, IETF, 2011, doi:10.17487/RFC6335.
[3] S. Sen, O. Spatscheck, D. Wang, Accurate, scalable in-network identification of P2P traffic using application signatures, in: Proceedings of the 13th International Conference on World Wide Web, ACM, 2004, pp. 512–521, doi:10.1145/988672.988742.
[4] T.T.T. Nguyen, G. Armitage, A survey of techniques for internet traffic classification using machine learning, IEEE Commun. Surv. Tut. 10 (4) (2008) 56–76, doi:10.1109/SURV.2008.080406.
[5] B. Li, J. Springer, G. Bebis, M.H. Gunes, A survey of network flow applications, J. Netw. Comput. Appl. 36 (2) (2013) 567–581, doi:10.1016/j.jnca.2012.12.020.
[6] H. Kim, K.C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, K. Lee, Internet traffic classification demystified: myths, caveats, and the best practices, in: Proceedings of the 2008 ACM CoNEXT Conference, ACM, 2008, pp. 11:1–11:12, doi:10.1145/1544012.1544023.
[7] M. Canini, W. Li, M. Zadnik, A.W. Moore, Experience with high-speed automated application-identification for network-management, in: Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ACM, 2009, pp. 209–218, doi:10.1145/1882486.1882539.
[8] M. Roughan, S. Sen, O. Spatscheck, N. Duffield, Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification, in: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement, ACM, 2004, pp. 135–148, doi:10.1145/1028788.1028805.
[9] A.W. Moore, D. Zuev, Internet traffic classification using Bayesian analysis techniques, SIGMETRICS Perform. Eval. Rev. 33 (1) (2005) 50–60, doi:10.1145/1071690.1064220.
[10] T. Auld, A.W. Moore, S.F. Gull, Bayesian neural networks for internet traffic classification, IEEE Trans. Neural Netw. 18 (1) (2007) 223–239, doi:10.1109/TNN.2006.883010.
[11] A. Callado, J. Kelner, D. Sadok, C.A. Kamienski, S. Fernandes, Better network traffic identification through the independent combination of techniques, J. Netw. Comput. Appl. 33 (4) (2010) 433–446, doi:10.1016/j.jnca.2010.02.002.
[12] A. McGregor, M. Hall, P. Lorier, J. Brunskill, Flow clustering using machine learning techniques, Passive Active Netw. Meas. (2004) 205–214, doi:10.1007/978-3-540-24668-8_21.
[13] S. Zander, T. Nguyen, G. Armitage, Automated traffic classification and application identification using machine learning, in: The IEEE Conference on Local Computer Networks 30th Anniversary, IEEE, 2005, pp. 250–257, doi:10.1109/LCN.2005.35.
[14] J. Erman, M. Arlitt, A. Mahanti, Traffic classification using clustering algorithms, in: Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data, ACM, 2006, pp. 281–286, doi:10.1145/1162678.1162679.
[15] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, C. Williamson, Semi-supervised network traffic classification, ACM SIGMETRICS Perform. Eval. Rev. 35 (1) (2007) 369–370, doi:10.1145/1269899.1254934.
[16] J. Zhang, X. Chen, Y. Xiang, W. Zhou, J. Wu, Robust network traffic classification, IEEE/ACM Trans. Netw. 23 (4) (2015) 1257–1270, doi:10.1109/TNET.2014.2320577.
[17] A. Dainotti, A. Pescape, K.C. Claffy, Issues and future directions in traffic classification, IEEE Netw. 26 (1) (2012) 35–40, doi:10.1109/MNET.2012.6135854.
[18] W. Li, M. Canini, A.W. Moore, R. Bolla, Efficient application identification and the temporal and spatial stability of classification schema, Comput. Netw. 53 (6) (2009) 790–809, doi:10.1016/j.comnet.2008.11.016.
[19] A. Fahad, Z. Tari, I. Khalil, I. Habib, H. Alnuweiri, Toward an efficient and scalable feature selection approach for internet traffic classification, Comput. Netw. 57 (9) (2013) 2040–2057, doi:10.1016/j.comnet.2013.04.005.

[20] J. Park, H.-R. Tyan, C.-C.J. Kuo, GA-based internet traffic classification technique for QoS provisioning, in: 2006 International Conference on Intelligent Information Hiding and Multimedia, IEEE, 2006, pp. 251–254, doi:10.1109/IIH-MSP.2006.264991.
[21] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, K. Salamatian, Traffic classification on the fly, ACM SIGCOMM Comput. Commun. Rev. 36 (2) (2006) 23–26, doi:10.1145/1129582.1129589.
[22] T.T.T. Nguyen, G. Armitage, P. Branch, S. Zander, Timely and continuous machine-learning-based classification for interactive IP traffic, IEEE/ACM Trans. Netw. 20 (6) (2012) 1880–1894, doi:10.1109/TNET.2012.2187305.
[23] F. Murai, B. Ribeiro, D. Towsley, P. Wang, On set size distribution estimation and the characterization of large networks via sampling, IEEE J. Sel. Areas Commun. 31 (6) (2013) 1017–1025, doi:10.1109/JSAC.2013.130604.
[24] J. Ma, K. Levchenko, C. Kreibich, S. Savage, G.M. Voelker, Unexpected means of protocol inference, in: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, ACM, 2006, pp. 313–326, doi:10.1145/1177080.1177123.
[25] J. Zhang, C. Chen, Y. Xiang, W. Zhou, Y. Xiang, Internet traffic classification by aggregating correlated Naïve bayes predictions, IEEE Trans. Inf. Forensics Secur. 8 (1) (2013) 5–15, doi:10.1109/TIFS.2012.2223675.
[26] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, Y. Guan, Network traffic classification using correlation information, IEEE Trans. Parallel Distrib. Syst. 24 (1) (2013) 104–117, doi:10.1109/TPDS.2012.98.
[27] Y. Wang, Y. Xiang, J. Zhang, W. Zhou, G. Wei, L.T. Yang, Internet traffic classification using constrained clustering, IEEE Trans. Parallel Distrib. Syst. 25 (11) (2014) 2932–2943, doi:10.1109/TPDS.2013.307.
[28] H. Yu, Y. Zhao, G. Xiong, L. Guo, Z. Li, Y. Wang, POSTER: mining elephant applications in unknown traffic by service clustering, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2014, pp. 1532–1534, doi:10.1145/2660267.2662391.
[29] Y. Wang, Y. Xiang, J. Zhang, W. Zhou, B. Xie, Internet traffic clustering with side information, J. Comput. Syst. Sci. 80 (5) (2014) 1021–1036, doi:10.1016/j.jcss.2014.02.008.
[30] D.R. Kerr, B.L. Bruins, Network Flow Switching and Flow Data Export, 2003, US Patent 6590894.
[31] J. Postel, RFC 792: Internet Control Message Protocol, Technical Report, Network Working Group, 1981, doi:10.17487/RFC0792.
[32] T. Fawcett, An introduction to roc analysis, Pattern Recognit. Lett. 27 (8) (2006) 861–874, doi:10.1016/j.patrec.2005.10.010.
[33] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32, doi:10.1023/A:1010933404324.
[34] N.V. Chawla, N. Japkowicz, A. Kotcz, Editorial: special issue on learning from imbalanced data sets, ACM SIGKDD Explorations Newslett. 6 (1) (2004) 1–6, doi:10.1145/1007730.1007733.
[35] C. Estan, G. Varghese, New directions in traffic measurement and accounting: focusing on the elephants, ignoring the mice, ACM Trans. Comput. Syst. 21 (3) (2003) 270–313, doi:10.1145/859716.859719.
[36] D. Tammaro, S. Valenti, D. Rossi, A. Pescapé, Exploiting packet-sampling measurements for traffic characterization and classification, Int. J. Netw. Manage. 22 (6) (2012) 451–476, doi:10.1002/nem.1802.



Lei Ding received his B.S. degree from Shandong University, Jinan, China, in 2005, and the M.S. degree from Xi'an Jiaotong University, Xi'an, China, in 2008, both in computing mathematics. He was a software engineer with Dazhong Daily News Group, Jinan, China, from 2008 to 2012. He is currently pursuing the Ph.D. degree in computer science at Xi'an Jiaotong University, Xi'an, China. His research interests include network traffic classification, network traffic modeling and analysis.

Jun Liu received his B.S. and Ph.D. degrees in computer science and technology from Xi'an Jiaotong University in 1995 and 2004, respectively. He is currently a professor of the Department of Computer Science and Technology at Xi'an Jiaotong University in China. His current research focuses on data mining and text mining and he has authored more than seventy research papers in various journals and conference proceedings. He served as a Guest Editor for many technical journals, such as Information Fusion, IEEE Systems Journal, and Future Generation Computer Systems. He also acted as a conference/workshop/track chair at numerous conferences.

Tao Qin received his B.S., M.S. and Ph.D. degrees in information engineering from Xi'an Jiaotong University, Xi'an, China, in 2004, 2006, and 2010 respectively. He is currently an associate professor with the Department of Computer Science and MoE KLINNS Laboratory of Xian Jiaotong University. His research interests include internet traffic analysis, abnormal detection and traffic modeling.

Haifei Li received his B.S. degree from Xi'an Jiaotong University, Xi'an, China, in 1990, and the M.S. and Ph.D. degrees from the University of Florida, Gainesville, FL, USA, in 1998 and 2001, respectively, all in computer science. He is currently an associate professor of Computer Science at Union University, Jackson, TN, USA. He has balanced academic/industrial experiences. He was an assistant professor of Computer Science at Nyack College from 2003 to 2004. He was a post-doctoral researcher at IBM Thomas J. Watson Research Center from 2001 to 2003. He was a graduate student at the University of Florida from 1996 to 2001. He was a software engineer at China Resources Information Technology from 1994 to 1996. He was a software engineer at Geophysical Research Institute from 1990 to 1994. He has published over 30 articles in various journals and magazines. His research interests are elearning, database, e-commerce, automated business negotiation and business process management.