Application Behavior Characterization (ABC) System for Fast and Accurate Large-Scale Traffic Classification

Dawei Wang[1,3], Luoshi Zhang[4], Zhenlong Yuan[1,2], Yibo Xue[1,3] and Yingfei Dong[5]
[1]*Research Inst. of Info. & Tech., Tsinghua University, Beijing, China*
[2]*Department of Automation, Tsinghua University, Beijing, China*
[3]*Tsinghua National Lab for Information Sci. & Tech., Beijing, China*
[4]*Computer Science & Technology College, Harbin Univ. of Sci. & Tech, Harbin, China*
[5]*Department of Electrical Engineering, University of Hawaii, Honolulu, HI 96822, USA.*
stonetools2008@gmail.com

***Abstract*—Network traffic classification is critical to both network management and system security. However, existing traffic classification techniques become less effective as more distributed applications (e.g., p2p systems) use proprietary protocols for delivery and encryption. Especially, current techniques usually focus on individual flows and do not consider all flows associated with an application together. To address this issue, in this paper, we propose a novel Application Behavior Characterization (ABC) System. We design a novel application behavior feature extracting method and an effective classification algorithm, which explore the correlation of multiple flows of a specific application. Furthermore, with a hierarchical architecture and a corresponding trigger mechanism, the proposed system can effectively achieve high performance with good efficiency. We implement a prototype system and evaluate with real network traffic. The experimental results show that it can first identify all flows belonging to a set of known distributed applications (such as Skype, Thunder, and PPTV) with over 90% precision and recall. Moreover, it can further identify one particular application that a flow belong to with 90% precision and recall on average, i.e., if a flow is generated by Skype, or Thunder, or PPTV.**

Keywords-Traffic Classification Systems, Distributed collaborative Protocol, Application Behavior, Decision Tree

I. INTRODUCTION

Real-time traffic classification is critical to both network management (such as Quality of Service) and system security (such as Intrusion Detection Systems, IDSs) [1]. Internet Service Providers (ISPs) often perform dynamic adaption and response based on traffic classification [2], e.g., ensure the quality of critical applications by limiting non-critical ones. They also use traffic classification for security analysis to detect various attacks [3][4][5].

Due to its broad use, traffic classification has been extensively examined in recent years, and many techniques have been proposed, including port-based techniques, payload-based techniques [7][8], and flow-based techniques [1]. However, new distributed applications with collaborative protocols among multi-parties (e.g., proprietary peer-to-peer, P2P, or peer-server hybrid systems) make existing techniques less effective due to the following reasons: (1) As many new applications use *random port numbers* instead of fixed well-known port numbers, port-based techniques do not work for them. (2) As many new application use *proprietary encryption protocols*, common payload-based techniques are inapplicable to their traffic. (3) Many new distributed applications use P2P or peer-server hybrid protocols to maintain services via *multiple collaborative flows*. These flows often use both TCP and UDP, and even modified version of TCP/UDP for both control and data communications. Because traditional flow-based techniques mostly focus on individual flows, they usually do not consider multiple flows together and miss the correlation among multiple flows from the same application.

Therefore, identifying traffic generated from specific applications (especially more and more new applications) is still a challenging issue. In this paper, we propose a novel Application Behavior Characterization (ABC) system which identifies traffic based on behavior features extracted from a set of flows belonging to certain applications. One key goal of the proposed system is to identify all flows associated with a specific application, which cannot be achieved by existing schemes. Main contributions of this paper include: (1) *We have developed a behavior-based traffic classification system*, which can effectively achieve high performance with good accuracy, as demonstrated in our experimental system. (2) *We have designed a novel application behavior feature extracting method*. Different from traditional flow features extracted from individual flows, we extract application behavior features from multiple flows associated with an application, for exploring the correlation among these flows. (3) *We have developed an effective classification algorithm*, which trains a two-layer machine learning model to classify application traffic. It helps us identify all flows associated with a specific distributed application. (4) *We have also designed an efficient double-table data structure*, for aggregating flows into clusters and extracting application behavior features more efficiently.

The reminder of this paper is as follows. We will introduce related work in Section II, and present the system design in Section III. We will discuss the proposed algorithm in Section IV and present our experimental evaluation in Section V. We will conclude this paper and discuss future work in Section VI.

*Corresponding author is Prof. Yibo Xue. E-mail: yiboxue@tsinghua.edu.cn.

II. RELATED WORK

A. Traffic classification

Traffic classification has been evolving along with the development of Internet protocol as shown in Fig. 1. Port-based techniques are the earliest methods that classify traffic based on well-known port numbers. However, they cannot deal with applications using random port numbers (e.g., Thunder), as it assumes that most applications consistently use "well known" TCP or UDP port numbers [6].

To address the issue of random port numbers, researchers developed payload-based techniques that classify traffic by comparing packet payloads with known signatures. These techniques usually use two matching methods: string-based matching and regular-expression matching. An example of string-based matching is OpenDPI, which support many protocols such as known P2P protocols, HTTP, and so on [9]. String-based matching can use fast multi-pattern matching algorithms but with limited expressivity [8]. To address this issues, regular expressions are used in later payload-based approaches [10][11]. Although payload-based techniques can classify traffic with known signatures very well, their effectiveness quickly diminishes when packet payloads are encrypted [7], especially when encrypted with proprietary protocols, such as Thunder or Skype.

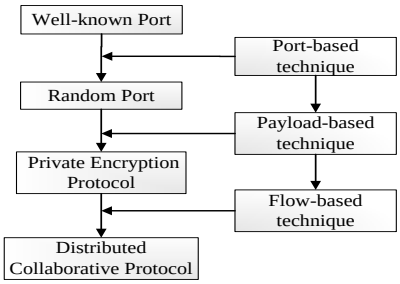

Figure 1. Evolution of application protocols and traffic classification.

To address the challenge of (proprietary) encryption protocols, researchers have developed flow-based techniques, exploring flows statistics instead of packet payload patterns. Most flow-based technique uses Machine Learning (ML) techniques as their classification algorithms. In 2004 McGregor et al. [12] published one of the earliest methods that applied ML in traffic classification using Expectation Maximization algorithms. Then many other approaches are proposed based on ML, such as Bayesian methods [13][14] and Support Vector Machine [15][16].

Based on the assumption that flows generated by different application protocols have unique statistical characteristics, flow-based techniques can identify the protocol that a flow belongs to, without inspecting packet payloads. However, exiting flow-based techniques do not explore the correlation among flows generated from the same application. First, many distributed applications use both TCP and UDP as their transport layer protocols at the same time. Because most existing flow-based techniques focus on either TCP or UDP flows, they cannot deal with multiple transport protocols simultaneously. Second, many distributed applications employ multiple flows to obtain services from several servers or peers. Existing flow-based techniques do not explicitly explore such information. One focus of our research is to address this issue.

B. Distributed Applications

Distributed applications (such as popular P2P or hybrid video streaming, voice-over-IP, real-time communications, content-sharing systems) generate most traffic on the current Internet. How to identify them becomes an important topic.

Skype [17] is a typical P2P VoIP application [19]. As it uses a proprietary protocol for communication and encryption, identifying Skype traffic is still a difficult task. Skype employs multiple flows to accomplish system operations: (1) It employs TCP flows for signaling, and both TCP and UDP flows for transporting media traffic. Signaling and transporting media traffic are not sent on the same ports [18]. (2) Relying on its P2P infrastructure to exchange data, it needs to keep backup nodes to ensure communications in the case of node failures. Thus, when a data exchange has be started, it still connects to many servers/peers.

Several methods have been proposed to classify Skype traffic. In [20], the authors focused on the relayed traffic generated by a Skype client, and developed a flow-based method to classify Skype traffic. Bonfiglio et al. designed a framework based on Chi-Square test and stochastic characterization to identify Skype voice traffic [21]. Rossi et al. focused on Skype signal traffic, and built a simple classification method for isolating different components of signaling activities related to various tasks [22]. In a whole operational cycle, a Skype client generates several different types of flows. However, these existing methods mainly concentrate on one type of Skype traffic, such as voice or signaling. They have not considered all Skype flows together as we do.

BitTorrent (BT) and its many variants present the same challenge. Because more and more BT applications use dynamic port numbers, masquerading techniques, and proprietary encryptions to avoid detection, identifying BT traffic also becomes increasingly difficult. Erman et al. used clustering algorithms to classify P2P BT traffic [23]. Karagiannis et al. proposed the BLINC method that classifies P2P traffic using host behavior at the transport layer [24]. These two methods, however, can only discover P2P-like traffic but cannot identify the specific protocol that the traffic belongs to. Aiming at a specific BT variant, Thunder, Heuristic Message Clustering technique (HMC) [25] is designed to obtain a state machine for characterizing communication behaviors of Thunder. However, it may be influenced by network noise. Because BT uses multiple flows to obtain service distributed across many servers/peers, and the statistical characteristics of each flow are different, the current flow-based techniques cannot find out all flows generated from a BT client. In this paper, we propose a novel classification framework to address this issue.

III. APPLICATION BEHAVIOR CHARACTERIZATION (ABC) SYSTEM

The proposed system is designed for identifying all flows associated with distributed applications. Different from current individual-flow-based systems, our system classifies traffic based on aggregated application behaviors. Here, by application behaviors, we refer to the relationships among flows generated by an application. We aggregate these flows into a cluster, extract application behavior features from this cluster, and determine the application that the flow cluster belongs to by feeding the features into classification models.

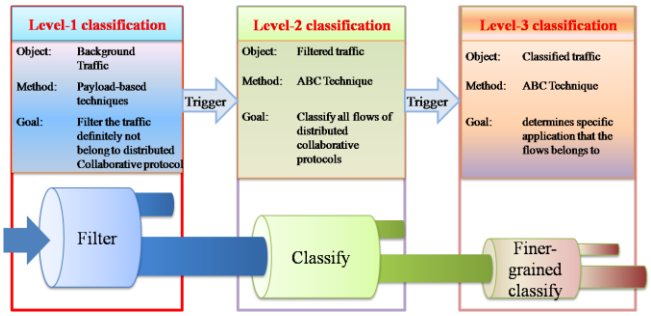

Figure 2. Overview of ABC System.

Fig.2 shows the overview of ABC system.

Hierarchical architecture. The proposed system uses a three-level hierarchical architecture, as shown in Fig.3. Level-1 classification filters traffic definitely not belonging to distributed applications, such as DNS, HTTP, or FTP flows. Level-2 classification determines if the remaining traffic belongs to distributed applications or not. Level-3 classification determines specific applications that the flows belong to.

Trigger mechanism. We use a trigger mechanism to link three classification levels. Two triggers are deployed in Level-1 and Level-2 classification, as shown in Fig.3. Traffic is not passed to the higher level until a corresponding trigger is activated based on the characteristics of specific distributed applications.

Parallel processing. We fully exploit multi-core processors to deal with the rapid increase of traffic volume to achieve high system performance.

A. System Artitechture

Fig.3 illustrates the proposed system architecture. As other typical traffic classification systems, we assume that all ingress/egress traffic for a domain (e.g., an ISP) goes through a gateway router. This gateway mirrors all traffic as the input for our system. Our system uses four modules to classify traffic from various distributed applications for further actions such as network management or security response:

- *Preprocessing module.* This module is the basis of the entire system. It captures and preprocesses packets, and dispatches preprocessed packets into asynchronous queues for parallel processing. Asynchronous queues are used to pass messages between processing threads.
- *Level-1 classification module.* This module filters out traffic that is definitely not belong to target applications using payload based classifiers. It also extracts flow features from unclassified traffic. These flow features are fed to the level-2 classification module as soon as the level-2 trigger is activated.
- *Level-2 classification module.* It is the core module of our system, used to identify distributed application traffic based on application behavior features. Besides the core functions, it also provides a trigger to determine when the classified application traffic needs to be passed to the level-3 module for finer-grained classification.
- *Level-3 classification module.* This module further identifies the specific applications that the flows belong to, such as Skype or Thunder.

For parallel processing, every core instantiates its own instances of level-1, level-2, and level-3 modules. The parallel processing starts as soon as the preprocessing module dispatches packets into asynchronous queues. We use a flow table and a *double table* to share data among the processing cores. A *flow table* maps packets with the same 5-tuple (protocol, source IP, source port, destination IP and destination port) into a flow. We introduce the details of double table in the following.

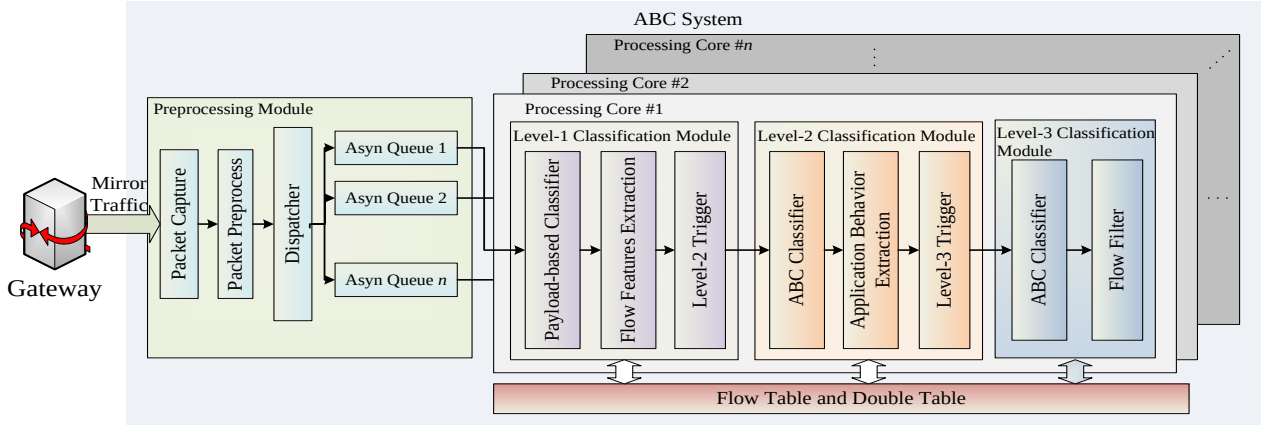

Figure 3. Architecture of ABC System.

B. *Double Table*

We will classify distributed application traffic based on application behavior features extracted from a cluster of flows. Because the flow table simply hashes individual flows based on their 5-tuples, it is difficult to aggregate related flows into a cluster. Therefore, we propose a *Double Table* to aggregate flows generated or received by a host into a cluster. The structure of Double Table is shown in Fig.4. For ease of discussion, we assume only one application runs on a host.

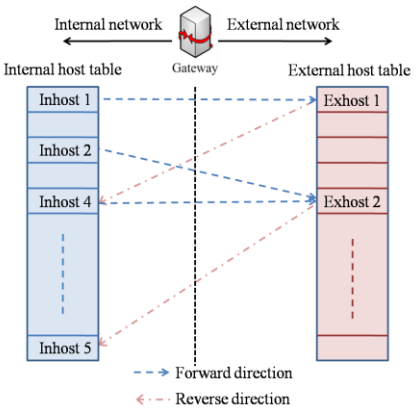

Figure 4. Structure of Double Table.

A double table is made of two separate tables: an internal host table and an external host table. The internal host table aggregates the flows generated or received by the hosts inside the local domain that we are monitoring, while the external host table stores the flows generated by external hosts. For example, "Inhost 4" in the internal host table aggregates the flows of sending to "Exhost 2" and sending from "Exhost 1".

Both internal/external tables are realized with hash tables and linked lists. The hash key is calculated using the IP address of an internal or external host. If collisions happen, the double table uses a linked list to store the collided hosts. Here, we further define that the flows started by internal hosts as the *forward direction*, and those started by external hosts as the *reverse direction*.

We use the double table to effectively aggregate the flows generated or received by a host into a cluster, and discover the collaboration among flows with the same application: (1) *Collaboration between TCP and UDP flows*. The flows aggregated in the double table for an internal or external host often contain both TCP and UDP flows. Thus, we can figure out how different applications use TCP and UDP differently in their communications. (2) *Collaboration among flows for different functions*. All flows generated by a application are aggregated in the double table. Thus, it helps easily figure out the collaboration among flows of an application for accomplishing different functions.

C. *Trigger mechanism*

We use two triggers to link three classification modules: one is between level-1 and level-2 modules; the second is between level-2 and level-3 modules. Packets are not passed to the higher level module until a trigger is activated. Because the trigger activated conditions are based on the characteristics of specific distributed applications, we can filter out traffic not belong to these applications. Only traffic meeting these conditions is processed.

Fig.5 is the flow chart of the preprocessing and the level-1 module. The main responsibility of level-1 classification module is to identify the traffic definitely not belonging to specific applications, such as HTTP, FTP, and DNS flows, using payload-based techniques. If a flow cannot be classified, the level-1 module checks the double table to find out whether the trigger of a internal host has been activated. We use two trigger conditions here: (1) *Unestablished TCP flows*. Most distributed applications attempt to connect peers in many other domains. Due to the dynamics of these applications, some peers are not alive anymore. Therefore, we can see many unestablished TCP flows during the life cycle of an application instance. (2) *Reverse direction UDP flows*. An instance of a distributed application is usually both a client and a server. As a server, it receives a reverse direction connection from a external host. It often uses UDP to generate reverse direction flows. Therefore, having reverse direction UDP flows is another trigger condition.

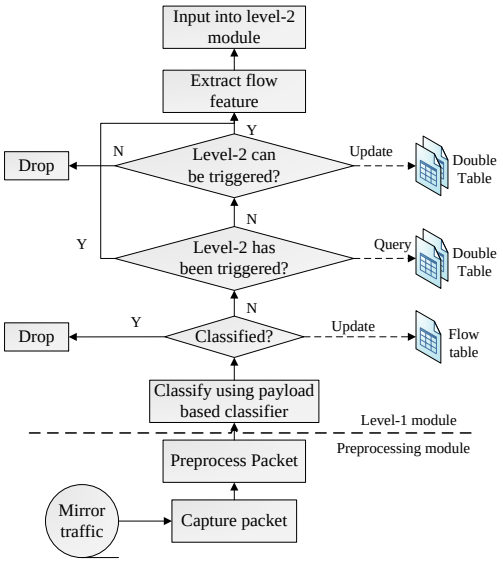

Figure 5. Flow chart of preprocessing and leve-1 module.

As soon as an internal host meets one of the above two conditions, the level-1 module starts to extract the features of the flows which are generated or received by a host but not yet classified by payload-based classifiers. The flow features considered here are: (1) the number of packets of a flow; (2) the number of bytes of a flow. These futures are passed to the level-2 module for further processing.

Fig.6 is the flow chart of level-2 module and level-3 module. The level-2 module is responsible for identifying all flows of a distributed application at a host, e.g., these flows are P2P flows. If fine-grained classification is required, we further define the trigger conditions after level 2 processing

to figure out it is Skype flows or Thunder flows. Traffic meeting these conditions will be passed to level 3 for further classification.

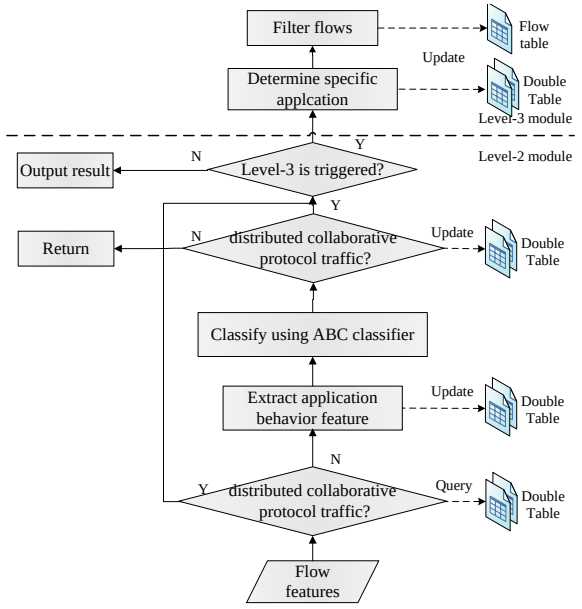

Figure 6. Flow chart of leve-2 module and leve-3 module.

In summary, our system design has the following advantages: (1) In the hierarchical architecture a lower level module narrows down the processing scope of a higher level modules. Therefore, the higher level modules can achieve a better performance. (2) Application-behavior-based triggers help us only focus on the highly-likely traffic to process. (3) We explore multi-core processors to meet practical challenges and achieve high performance.

IV. PPLICATION BEHAVIOR CHARACTERIZATION TECHNIQUE

We develop an Application Behavior Characterization technique, consisting of two main parts: (1) An *application behavior feature extracting method*, for extracting application behavior features from multiple flows. (2) A *two-layer classification algorithm* based on C4.5 decision trees. This classifier fully takes advantage of application behavior features to classify application traffic. We deploy this technique in the level-2 module.

C4.5 is a multi-class classification algorithm, and is variant of decision tree learner [26]. It builds decision trees from a set of training data using information entropy [27]. The decision tree is a recursive structure including: (1) *leaf* nodes labeled with a class value; (2) *test* nodes that have two or more outcomes and each linked to a sub-tree. At each node, C4.5 chooses one attribute of the data that most effectively splits its sample set into subsets enriched in one class or the other.

A. *Extracting Application Behavior Features*

Flow-based techniques are less effective when they are used to classify traffic from distributed applications, such as P2P systems. The key reason is that the flow features obtained from a single flow are not suitable for identifying distributed applications. Therefore, we propose to extract application behavior features from multiple flows to exploit the correlation among flows for effective classification.

The proposed method extracts behavior features from multiple flows generated or received by an internal host. Noted that a distributed application tends to use both TCP and UDP as their transport protocols. So we extract application behavior features from TCP and UDP flows separately.

For TCP flows, the extracting method divides them into three groups. (1) *Unestablished TCP flows (T1)*. During the life cycle of application instance, it usually generates a few unestablished TCP flows. We consider these flows as an important characteristic of distributed applications. Technically, the TCP flows that only have SYN packets from internal hosts are taken as the unestablished ones. (2) *Short TCP flows (T2)*. Most distributed applications connect to some servers before they start to transmit application data. These flows tend to be short as they do not carry any application data. As we observed the volume of distributed application traffic, we found that there are a lot of flows contain less than 15 packets. Therefore, the TCP flows that have less than 15 packets are taken as short TCP flows. (3) *Long TCP flows (T3)*. This type of flows is used to transmit application data, so they contain many packets. The TCP flows that contain more than 15 packets are taken as long TCP flows.

For UDP flows, the extracting method divides them into two groups. (1) *UDP flows of forward direction (U1)*. These flows are used to require application data from other peers. (2) *UDP flows of reverse direction (U2)*. These flows which are used to response the requirement of application data from other peers, are another important characteristic of distributed applications.

The application behavior feature can be stated as

$$<T1, T2, T3, U1, U2> \quad (1)$$

For each group of both TCP and UDP flows, the proposed method extracts four features, shown in table I. Then the features of each group is a 4-dimensional vector,

$$<num\_outside\_ip, num\_flow, num\_packets, num\_bytes> \quad (2)$$

TABLE I. FETURES OF EACH GROUP

No.	*Name*	*Description*
1	num_external_ip	Number of external IP addresses that an internal host connects to in a group
2	num_flow	Number of flows that an internal host generates or receives in a group
3	num_packets	Number of packets of the flows in a group
4	num_bytes	Number of bytes of the flows in a group

Finally, the application behavior feature is a 20-dimensional vector.

Fig.7 illustrates the extracting process of application behavior features. In this method, flow features from the level-1 module are classified into two types: (1) *regular flow features*, including T2, T3, and U1; (2) *trigger flow features*, including T1 and U2. As the first trigger flow feature is popped up from an input queue, the first application behavior feature vector is created for an internal host. Meanwhile, the trigger flow features are also used to update a trigger pool. The extracting method maintains a trigger pool for every feature vector, and presets a initial size ρ_t for them. If the trigger pool of the last feature vector is full, a new feature vector is created. For example, in Fig.7, we preset ρ_t to 5. The trigger pool of *feature vector 2* has a size of 4. As a new trigger flow feature comes, the trigger pool is full to 5. Then, a new feature vector (feature vector 3) is created. Both regular flow features and trigger flow features are used to update all the behavior feature vectors. In addition, the double table also maintains a sliding window for every host, as shown in Fig.8. The feature vector whose trigger pool is full will be moved into the sliding window. Meanwhile, The double table only maintains a preset number of application behavior feature vectors for a host. As soon as the number of the current vectors exceeds the preset number, the extracting method will stop updating the first one and delete it.

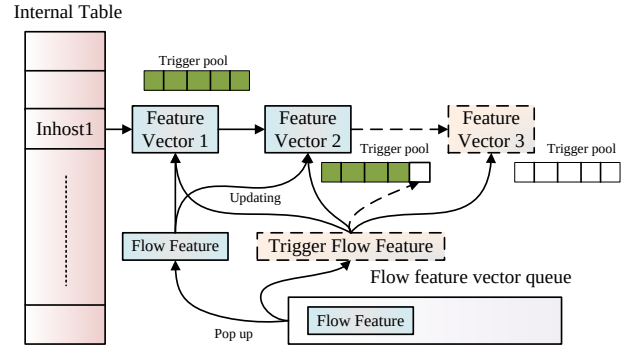

Figure 7. Extracting application behavior features.

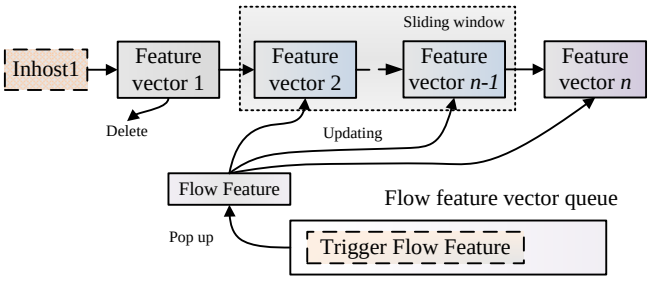

Figure 8. Sliding window.

After extracting, the application behavior feature vectors are passed to the two-layer classification algorithm for classification. This algorithm is divided into two stages: training stage and classification stage.

B. *Training stage*

Before training, we first extract the training set from a set with pure target application traffic. When extracting, we will add all application behavior feature vectors in a sliding window into the training set as soon as the first feature vector in the window is deleted. Then all the vectors in the sliding window are deleted from the double table, and the new training feature vectors are extracted. By this way, all the possible features of application behavior can be extracted for training.

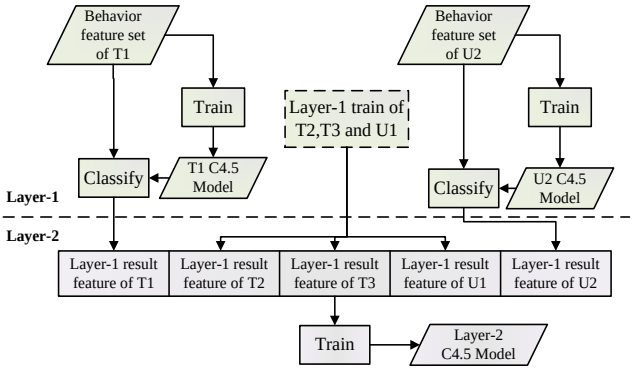

Figure 9. Training stage.

Fig.9 shows the training stage. In the first layer training, five C4.5 decision trees are trained for every group. When training these decision trees, the vectors of each group whose attributes are all zero are filtered. After training, these five C4.5 decision trees are used to classify the training set respectively, and calculate a two-dimensional vector, stated as

$$<class, error\ rate> \tag{3}$$

where the attribute *class* is the type that a first layer decision tree classifies a vector to be, and the attribute *error rate* is the rate that the vector is classified into this *class* by error. The error rate can be calculated by

$$error\ rate_{class} = 1 - \frac{N_{class}}{\sum_{i=1}^{n} N_i} \tag{4}$$

where N_{class} is the number of vectors in the training dataset that are classified to *class* in a leaf node, and N_i is the number of each class of the training dataset in the leaf node. As a zero vector appears, the layer-1 training will use <*none*, *1.0*> instead of the classification result.

After the first layer training, the original 20-dimensional feature vector becomes a 10-dimensional feature vector named as a layer-1 result feature vector. The layer-1 result feature vectors are passed to the second layer for training. The second layer training also employs C4.5 to train a decision tree using a layer-1 result feature vector set. So far, the training stage finishes with five layer-1 decision trees and a layer-2 decision tree.

C. Classification stage

In the classification stage, all the feature vectors in the sliding window of a host are passed to the two-layer classification algorithm for classification, when the number of refreshments of them is more than a preset threshold ρ_d. Fig.10 demonstrates the classification using the two-layer classification model.

In the first layer classification, a 20-dimensional behavior feature vector is passed to five layer-1 C4.5 decision trees to calculate the classification results of each group separately. Also, the classification results of the zero vectors are represented by *<none, 1.0>*. After the first layer classification, we obtain a 10-dimensional layer-1 result feature vectors. Then these vectors are classified by the layer-2 C4.5 model. After the second layer classification, the final classification results are obtained.

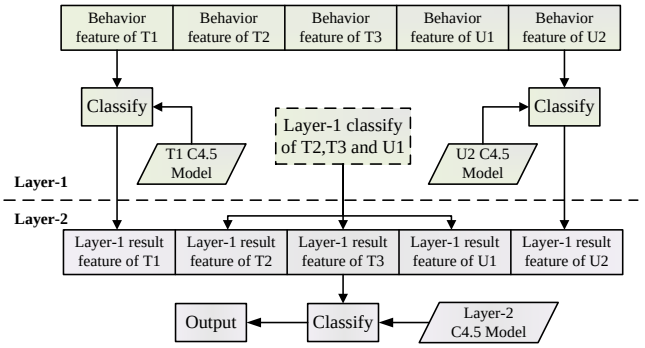

Figure 10. Classification stage.

If all the final classification results of the feature vectors with more than ρ_d flows in a sliding window indicate the current traffic does not belong to distributed applications, the classification result is ignored. Otherwise, the double table is updated according to the classification result as shown in Fig.11. If an application behavior feature in the sliding window is classified into certain distributed applications, the classification result is output and the feature vectors that have not been classified will be deleted.

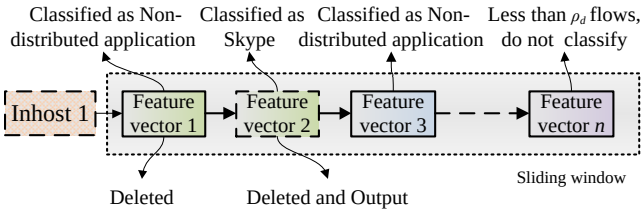

Figure 11. Updating the Double Table.

D. Algorithm analysis

The key differences between the proposed technique and other traffic classification techniques are in the feature extracting method and the classification algorithm. The extracting method extracts a new type of features from multiple flows—*application-level behavior feature*. The main advantages of the proposed system are: (1) Our application-level behavior feature extracting method integrates the statistical features of both TCP and UDP flows. Therefore, it can provide more information to classify distributed application traffic more precisely. Meanwhile, the integration can also help us address the *dynamic changes of transport protocol* in distributed applications. (2) Our behavior feature extracting method focuses on multiple flows instead of a single flow. Thus, using such application behavior, the proposed classification algorithm can identify all flows associated with distributed applications. (3) Our feature extracting method divides distributed application flows into five different groups. Due to the fact that different distributed applications may have different characteristics in different groups, the proposed algorithm has a better chance to determine the protocol used for the application traffic.

Based on the application behavior feature, a novel classification algorithm based on ML is used to classify application traffic. Using this two-layer classification algorithm, we can make better use of application behavior features. (1) The first layer processes the feature vectors of every group separately, which can fully take advantage of the grouping characteristic of application behavior features. (2) The second layer synthesizes the classification results of layer-1, and provides a final classification result based on the considerations of every group. Therefore, we can not only identify distributed application traffic, but also classify its associated application protocol.

In summary, our application behavior feature extracting method and two-layer classification algorithm can effectively identify distributed application traffic.

V. Performance Evuluation

A. Datasets and metrics

In this section, we present our experimental evaluation of the proposed system and classification technique. The experiments are carried out on four datasets:

- *Non-distributed dataset*. This dataset contains the non-distributed applications, e.g., HTTP, FTP, Online Game, point-to-point Instant Messaging (except Skype), and so on.
- *Skype dataset*. This dataset is generated with Skype clients in our lab.
- *Thunder dataset*. This dataset is generated by Thunder clients in our lab. Thunder is a popular BT variant in China.
- *PPTV dataset*. This dataset is generated with PPTV clients in our lab. PPTV is a popular P2P video streaming software.

We capture these four datasets at the gateway of our lab. When generating the data sets, all the hosts in our lab use the same protocol. For example, when we capture the dataset of PPTV, all the hosts use PPTV. When we capture the dataset of Non-distributed applications, all the hosts can uses any protocol but target distributed applications: Skype, Thunder, and PPTV.

Each dataset is divided into two separate sub-datasets: (1) training sub-dataset; (2) testing sub-dataset. The training sub-dataset is used to train our system, while the testing sub-dataset is used to test our system. The statistical information of these four datasets is listed in Table II.

TABLE II. STATISTICAL INFORMATION OF FOUR DATASET

Dataset		*Number of TCP flows*	*Number of UDP flows*
Training	***Non-distributed***	83,474	264,107
	Skype	56,428	151,479
	Thunder	176,381	675,531
	PPTV	3,294	547,444
Classification	***Non-distributed***	172,291	451,302
	Skype	199,083	653,732
	Thunder	219,474	1,405,242
	PPTV	14,425	1,876,533

We use two flow-level metrics to evaluate the proposed system: (1) *Recall* (*R*) represents the fraction of flows that are positively classified. (2) *Precision* (*P*) represents the fraction of predicted instances that are positively predicted.

We deploy our system on dual Intel Xeon E5504 processors (8-cores). One core runs the preprocessing module; six cores perform traffic classification; and the last core is responsible for maintaining the flow table and the double table.

B. *Parameter Selection*

The size of trigger pool ρ_t and classification threshold ρ_d are important parameters of the proposed system. Currently, we use empirical values to set them up. For different applications, different values are selected to balance the performance. The following experiments describe the effect of the ρ_t and ρ_d on the performance of our system. Furthermore, according to the traffic characteristics of distributed applications, the proper choices of ρ_t and ρ_d can be selected through these experiments.

First, we carry out the experiment to select the size of trigger pool ρ_t. At the training stage, we mix the four training datasets into one dataset to train the system. At the classification stage, we mix the four classification datasets to test the system. Fig.12 and Fig.13 show the trend of precision and recall, as we adjust ρ_t from 1 to 15. Fig.14 demonstrates the effect of ρ_t on the number of application behavior features. From these figures, we can see that both the precision and recall of our system remain at a relatively high level as ρ_t increases. As we increase ρ_t, the precision and recall increase a little, but the number of application behavior features is greatly decreased. As less features need to be processed, we can achieve higher performance. However, the larger ρ_t is, the longer the delay to classify traffic, because we need more trigger flow features to update the application behavior features when ρ_t is larger. Thus, consider classification effectiveness and efficiency together, we carry out the following experiments with ρ_t equal to 5.

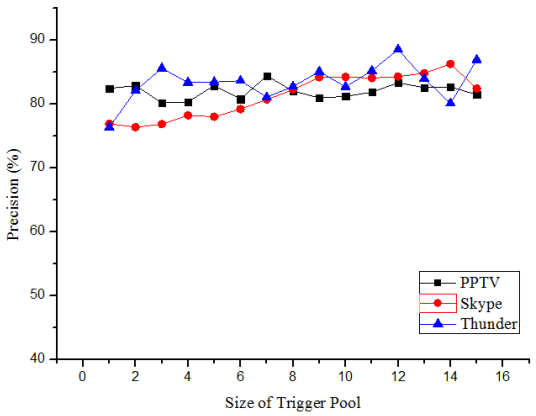

Figure 12. Precision trend with different ρ_t

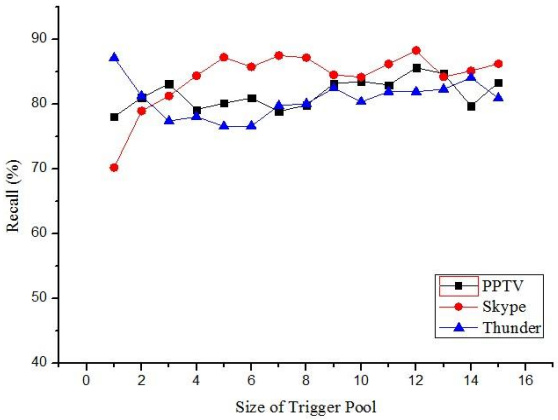

Figure 13. Recall trend with different ρ_t.

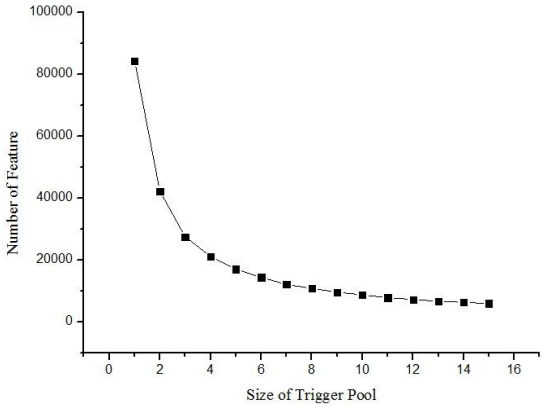

Figure 14. Number of features with different ρ_t.

The following experiment is designed to select classification threshold ρ_d. As same as the above experiments, we mix the four training datasets into one dataset to train system at the training stage, and mix the four classification datasets to test system at the classification stage. Fig.15 shows the average *Precision* of the classification results of three distributed applications. As shown in the figure, the

average *Precision* is relatively low when ρ_d is less than 8. As we adjust ρ_d to more than 8, the average *Precision* remains at a relatively high level. However, the larger ρ_d is, the longer the delay to classify traffic, because we need more flow features to update the application behavior features when ρ_d is lager. Thus, we carry out the following experiments with ρ_d equal to 8. Furthermore, this experiment also demonstrates that we can classify all flows associated with a specific distributed application precisely within 8 flows.

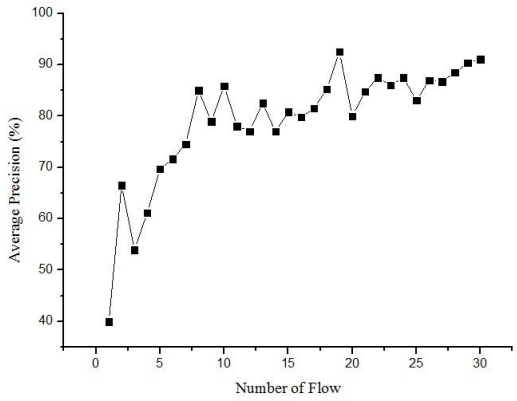

Figure 15. Precision trend with different ρ_d.

C. Performance of classifying distributed application traffic

We conduct these experiments to verify that the proposed system can classify all flows of target applications from background traffic.

At the training stage, we initialize three training datasets by mixing the Non-distributed training dataset with Skype, Thunder, and PPTV training dataset, respectively. Then we use these three training dataset to train system separately. At the classification stage, we also initialize three classification datasets by mixing the Non-distributed classification dataset with Skype, Thunder, and PPTV classification dataset. Then three classification datasets are passed to the system for testing. Furthermore, we mix the Skype, Thunder, and PPTV traffic as the *Distributed* traffic set, and further mix it with *Non-distributed* traffic to test our system. The experimental results are shown in Table III.

From Table III, we can see that the *Precision* results of *Non-distributed* and the *Recall* results of three distributed applications are relatively high. These results demonstrate that some flows of *Non-distributed* are classified as three distributed applications by mistake. The reason is that *Non-distributed* dataset contains many protocols, while the mixed traffic of these protocols may mislead the classification. Especially we test the system using *Non-distributed* and *Skype* datasets, more errors occurred. The best performance is the "*Non-distributed and distributed*". This is because, the more distributed traffic is mixed, the more knowledge provide to train the two-layer classification model. Furthermore, the *Recall* results show that over 90% flows of the distributed application traffic are positively classified. As the fact that the distributed application traffic contains different types of flows generated by distributed applications, the result of *Recall* proves that the proposed system can identify all flows associated with target applications.

TABLE III. EXPERIMENTAL RESULTS OF CLASSIFYING DISTRIBUTED COLLABORATIVE PROTOCOL TRAFFIC

Dataset	***Non-distributed***		***Skype, Thunder and PPTV***	
	Precision	***Recall***	***Precision***	***Recall***
Non-distributed and Skype	98.47%	73.65%	65.18%	97.72%
Non-distributed and Thunder	80.09%	67.65%	82.06%	90.25%
Non-distributed and PPTV	91.13%	73.52%	77.94%	92.90%
Non-distributed and distributed	90.07%	89.73%	92.18%	93.37%

D. Performance of identifying the specific applications

We now show the fine-grained classification to identify target applications. In these experiments, we mixed traffic from three application training datasets into one training dataset with three classes, and mixed all classification datasets into one classification dataset. Then we train a two-layer classification model, and test it with the mixed classification dataset. The experimental results are shown in Table IV and Table V.

TABLE IV. EXPERIMENTAL RESULTS OF IDENTIFYING PROTOCOL OF CLUSTER OF FLOWS

Application protocol	***Precision***	***Recall***
PPTV	93.66%	92.82%
Skype	91.01%	86.69%
Thunder	90.96%	95.73%

TABLE V. DETAIL RESULTS OF IDENTIFYING PROTOCOL OF CLUSTER OF FLOWS

Classified as / ***Application protocol***	***PPTV***	***Skype***	***Thunder***
PPTV	92.82%	3.88%	3.30%
Skype	6.80%	86.69%	6.51%
Thunder	1.05%	3.23%	95.73%

From Table IV, we can see that among the three protocols, we achieve the best *Precision* results with PPTV. This is because of the unique characteristics of PPTV comparing with other two applications. As a user demands a video, a PPTV client attempts to connect several servers using TCP flows first, and then the video data is transmitted via UDP flows. As long as the current transmission rate can guarantee real time streaming, the PPTV client will not connect to other servers/peers. Thus, the PPTV traffic contains less TCP flows comparing other applications. In summary, the experimental results shown in Table IV can prove that our system can identify target applications with high accuracy.

Table V shows the detailed classification results of three distributed applications. We achieve the best results with Thunder. Of all Thunder flows, 95.73% are classified as Thunder correctly, and 1.05% and 3.23% are classified as PPTV and Skype by mistake, respectively. This worst results are on Skype. Of all Skype flows, more than 10 % flows are mistakenly classified as PPTV and Thunder. This is because that Skype provides more services than Thunder and PPTV. Using Skype, a user can chat with his friend with text, voice or video. Skype also can transmit files between users. On the contrary, both Thunder and PPTV usually provide one service, which is file downloading or video streaming. In this case, certain application features of Skype may show the similar characteristics of PPTV or Thunder. Thus, the flows of these features are mistakenly classified.

VI. CONCLUSIONS

In this paper, we have presented the Application Behavior Characterization (ABC) system for identifying distributed applications from other traffic. We have then introduced a novel application behavior feature extracting method and an effective classification algorithm with corresponding analysis and experimental evaluation. The purposed system is able to identify all flows associated with specific applications and further determine the application that these flows are associated with. The experimental results show that we can achieve high performance on multi-core systems for practical online classification. We will further improve the proposed method to enhance the ability to distinguish the similar service provided by different applications, e.g., both Skype and Thunder support file transfer.

ACKNOWLEDGMENT

This work was supported by National High-Tech R&D 863 Program of China under grant No. 2010AA012502, and Tsinghua National Laboratory for Information Science and Technology (TNList) Cross-discipline Foundation.

REFERENCES

[1] T. Nguyen and G. Armitage, "A survey of techniques for Internet traffic classification using machine learning," IEEE Communications Surveys & Tutorials, Vol. 10, No.4, 2008, pp. 56-76.

[2] L. Bernaille, R. Teixeira, I. Akodkenou, "Traffic classification on the fly", ACM SIGCOMM Computer Communication Review, Vol. 36, No. 2, 2006, pp. 23-26.

[3] Snort - The de facto standard for intrusion detection/prevention, http://www.snort.org, as of August 14, 2007.

[4] Bro intrusion detection system - Bro overview, http://bro-ids.org, as of August 14, 2007.

[5] L. Stewart, G. Armitage, P. Branch, and S. Zander, "An architecture for automated network control of QoS over consumer broadband links," in preceedings of IEEE International Region 10 Conference (TENCON 05), Melbourne, Australia, November 2005.

[6] T. Karagiannis, A. Broido, N. Brownlee, and K. Claffy, "Is P2P dying or just hiding?" in proceedings of the 47th annual IEEE Global Telecommunications Conference, Dallas, Texas, USA,December 2004.

[7] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in network identification of P2P traffic using application signatures," in proceedings of WWW 2004, New York, NY, USA, May 2004.

[8] R. Smith, C. Estan, S. Jha and et al, "Deflating the big bang: fast and scalable deep packet inspection with extended finite automata," in proceedings of the ACM SIGCOMM 2008, Seattle, USA, August 2008.

[9] OpenDPI Integration Manual, http:// opendpi.googlecode.com/files/ OpenDPI-Manual.pdf, as of September 4, 2009.

[10] R. Sommer and V. Paxson, "Enhancing byte-levelnetwork intrusion detection signatures with context," In proceedings of the 10th ACM Conference on Computer and Communications Decurity (CCS 2003), Chicago, USA, October 2003.

[11] R. Smith, C. Estan, and S. Jha. "XFA: faster signaturematching with extended automata," In IEEE Symposium on Security and Privacy, Oakland, USA, May 2008.

[12] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques", in proceedings of Passive and Active Measurement Workshop (PAM2004), Antibes Juan-les-Pins, France, April 2004.

[13] A. Moore and D. Zuev, "Internet traffic classification using Bayesiananalysis techniques," in ACM International Conference on Measurementand Modeling of Computer Systems (SIGMETRICS) 2005, Banff, Alberta, Canada, June 2005.

[14] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian neural networks for Internet traffic classification," IEEE Trans. Neural Networks, no. 1, pp. 223–239, January 2007.

[15] B.H. Yang, G.D. Hou, L.Y. Ruan and et al. "SMILER: towards proactical online traffic classification". In proceedings of ANCS 2012.

[16] A. Este, F. Gringoli and L. Salgarelli. "Support vector machines for TCP traffic classification," Computer Networks, Vol.53, Issue 14, Septemper 2009.

[17] Skype web site, <http://www.skype.com>.

[18] S. A. Baset and H. Schulzrinne, "An analysis of the skype peer-to-peer Internet telephony protocol," in proceedings of Infocom 2006, Barcelona, Spain, April 2006.

[19] KaZaa web site. http://www.kazaa.com

[20] K. Suh, D. R. Figueiredo, J. Kurose and et al, "Characterizing and detecting Skype-relayed traffic," In proceedings of Infocom 2006, Barcelona, Spain, April 2006.

[21] D. Bonfiglio, M. Mellia and M. Meo, "Revealing Skype traffic: When randomness plays with you," In Proceedings of Sigcomm 2007, Kyoto, Japan, August 2007.

[22] D. Rossi, M. Mellia and M. Meo, "Understanding Skype signaling," Computer Networks. Vol. 53. Issue 2, 2009, pp.130-140.

[23] J. Erman, M. Arlitt and A. Mahanti, "Traffic classification using clautering algorithms". In proceedings of Sigcomm 2006 workshop, Pisa, Italy, September 2006.

[24] T. Karagiannis, K. Papagiannaki and M. Faloutsos. BLINC: Multilevel Traffic classification in the dark. In proceedings of Sigcomm 2007, Kyoto, Japan, August 2007.

[25] C.L. Li, Y.B. Xue and Y.F. Dong, HMC: a novel mechanism for identifying encrpyted P2P Thunder Traffic. In proceedings of Globecom 2010, Miami, USA, December, 2010.

[26] J. R. Quinlan. C4.5: Programs for Machine Learning. San Mateo: Morgan Kaufmann, 1993.

[27] S.B. Kotsiantis, Supervised Machine Learning: A Review of Classification Techniques, Informatica, Vol.31, 2007, pp. 249-268.