Computer Networks 53 (2009) 849–863

Contents lists available at ScienceDirect

Computer Networks

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

Profiling and identification of P2P traffic ☆

Yan Hu [a,*], Dah-Ming Chiu [a], John C.S. Lui [b]

[a] Department of Information Engineering, Chinese University of Hong Kong, Hong Kong
[b] Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong

ARTICLE INFO

Article history:
Available online 27 November 2008

Keywords:
Application Identification
Behavioral profile
Data mining

ABSTRACT

Accurate identification of network applications is important for many network activities. The traditional port-based technique has become much less effective since many new applications no longer use well-known fixed port numbers. In this paper, we propose a novel profile-based approach to identifying traffic flows belonging to the target application. In contrast to the method used in previous studies, of classifying traffic based on statistics of individual flows, we build behavioral profiles of the target application, which describe dominant patterns in the application. Based on the behavior profiles, a two-level matching method is used to identify new traffic. We first determine whether a host participates in the target application by comparing its behavior with the profiles. Subsequently, we compare each flow of the host with those patterns in the application profiles to determine which flows belong to this application. We demonstrate the effectiveness of our method on-campus traffic traces. Our results show that one can identify popular P2P applications with very high accuracy.

1. Introduction

The accurate identification of network applications is important in many areas such as network planning, quality of service (QoS) and access control. To identify network applications, the simplest method is to use transport-layer port numbers, since many services are supposed to run on well-known ports. Today, however, this traditional port-based technique has become less accurate for many reasons. For instance, some new applications, such as P2P applications, do not rely on predefined well-known ports. Another reason is that firewalls block some unauthorized or unknown applications, so some applications are tunneled through port 80 in order to circumvent firewalls. Another approach is payload-based analysis, which searches packet payloads for the signatures of known applications. This approach is employed in some commercial network management products. However, finding appropriate signatures and maintaining up-to-date signatures for various applications are daunting tasks. Moreover, the payload-based method is ineffective when traffic is encrypted.

Today, peer-to-peer applications have grown to represent a large proportion of Internet traffic. P2P traffic identification is especially important in network management since many P2P applications are bandwidth-intensive. Yet, P2P applications are even harder to identify than the traditional network applications because of their complexity. Furthermore, some newer-generation P2P applications are incorporating various strategies to avoid detection.

Recently, some novel approaches [9,19,21] have tried to classify network traffic by relying on statistical observations of the flows. These approaches develop discriminatory criteria based on statistical observations (such as the packet size distribution per flow, flow duration, and the statistics of the inter-arrival times between packets in flows, etc.), and then employ clustering, classification and other machine learning techniques to classify the traffic. These methods provide a promising alternative for traffic classification.

☆ A preliminary version of this paper appeared in the 16th International Workshop on Quality of Service (IWQoS) 2008.

* Corresponding author. Tel.: +86 15001312760.
E-mail addresses: yhu4@ie.cuhk.edu.hk (Y. Hu), dmchiu@ie.cuhk.edu.hk (D.-M. Chiu), cslui@cse.cuhk.edu.hk (J.C.S. Lui).

1389-1286/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2008.11.005

There are two challenges in identifying P2P applications using flow properties. First, different flows in the same application may have different flow statistics, since P2P applications are usually complicated. For example, in the P2P file-sharing applications, some flows are used to get peer information, other flows are to negotiate between peers, and other flows are involved in the actual file transfer. These various kinds of flows have different statistical properties and even different transport protocols. Second, some flows in the given application do not have obvious and specific flow statistics. If we only look at per-flow statistics, these flows are very similar to some flows in other applications.

In this paper, we propose a new approach to identify P2P applications that has three new properties. These three properties can improve the accuracy of identifying P2P applications.

- **Profile-based**. In addition to observing statistics of individual flows, we build *profiles* for a given application, which describe the communication patterns of this application. Comparing a host's behavior with the profiles is equivalent to looking at properties of multiple flows of this host, which contain more information than a single flow.
- **Two-level matching**. The first level determines whether a host participates in a given application by comparing its behavior with the profiles of this application. Since a host may take part in different applications at the same time, we then determine in the second level which flows belong to this application and which do not. Although flows in different applications may have similar statistics, two-level matching can reduce the rate of false positives.
- **Flow correlation**. Since some flows do not have peculiar per-flow statistics, we use correlation between these flows and other flows that have already been identified as belonging to the given application to reduce the rate of false negatives.

To build profiles for a given P2P application, the first step is to choose appropriate features to be used in the profiles. After that, we observe the properties of flows that are contained in the target application. Since manually extracting properties from large numbers of flows is difficult and time-consuming, we use a data mining technique called *association mining* in this step. We process the association rules to build application profiles and find correlations between flows satisfying different rules. Finally, in the application identification step, the matching is implemented at two levels, the host-level and the flow level.

The rest of the paper is organized as follows: we review the related work in Section 2. Before describing our approach in detail, we first give an example of application profiles in Section 3. After that, we explain how to build behavioral profiles for the target application in Section 4. In Section 5, we describe the two-level matching method to identify the P2P application in new traffic. Experimental results are given in Section 6. Section 7 discusses open issues, practical considerations and future work directions. Finally, we conclude our paper in Section 8.

2. Related work

Due to its wide application, the field of traffic classification has received continuous interest. Some traffic classification approaches develop discriminating criteria-based on statistical observations of various flow properties in the packet traces. Based on these statistical observations, these studies employ classification, clustering and other machine learning techniques to assign flows to classes. Roughan et al. in [21] classify traffic flows into four classes suitable for quality of service applications. They demonstrate the performance of nearest neighbor and linear discriminant analysis algorithms. Moore et al. in [19] apply Bayesian analysis techniques to categorize traffic by application.

In addition to these trained classification techniques, unsupervised clustering methods are also used in several studies. The work of Campos et al. in [11] applies a number of different hierarchical clustering methods presented as dendrograms to identify groups of similar communication patterns. Similarly, in [18], McGregor et al. seek to identify traffic with similar observable properties and apply a probabilistic clustering method (the EM algorithm) to this problem. Most of these studies that apply machine learning techniques focus only on the statistics of a single flow, while our approach observes the patterns of multiple flows.

Another promising traffic classification approach that is closer to our method is shown in [13]. Instead of classifying individual flows, Karagiannis et al. propose to associate Internet hosts with applications, and then classify their flows accordingly. They attempt to capture the inherent behavior of a host at three levels of increasing detail: the social, functional and application levels. This approach mainly focuses on higher level communication patterns such as the number of source ports a particular host uses for communication, and does not make use of flow statistics such as ours. Concurrent to [13], Xu et al. in [24] use information theoretic and data mining techniques to build behavior profiles of Internet backbone traffic.

In [5], Bernaille et al. evaluate the feasibility of application identification at the beginning of a TCP connection. This approach distinguishes the behavior of an application by observing the size and the direction of the first few packets of the TCP connection. Crotti et al. in [8] propose a statistical approach to identify network application by building a set of protocol fingerprints that summarize its main IP-level statistical properties. In [17], Ma et al. analyze three mechanisms relying on flow content to automatically identify traffic that uses the same application-layer protocol.

The stunning growth of P2P traffic has recently attracted increased attention from researchers. Several studies have emphasized the identification of P2P traffic, such as the signature-based payload methodology in [22] and the identification method by transport-layer characteristics in [12]. In [7], Collins et al. propose a set of tests for identifying masqueraded peer-to-peer file-sharing applications. Bonfiglio et al. in [6] propose a framework to reveal Skype traffic in real time by exploiting the randomness introduced by the encryption process at the bit level.

3. An example application profile

Before describing our approach in detail, we first give an example of application profiles in this section. We take a popular peer-to-peer file-sharing application, BitTorrent, as an example. The BitTorrent network consists of clients and a centralized server. Clients (called downloaders) connect to each other directly to send and receive pieces of a file. The central server (named tracker) coordinates the action of the downloaders. Instead of describing the protocol details, we focus here on the flow properties of communication between downloaders and between the downloader and the tracker. We also describe the corresponding payload signatures in different traffic, which are used in training and validation. However, the payload signatures are not contained in the application profiles.

There are roughly four kinds of traffic in a BitTorrent application: (1) users download the torrent files, (2) downloaders periodically check in with the tracker, (3) peers communicate with each other, and (4) communication occurs between DHT (Distributed hash table) nodes. To download a file, a user should first get the corresponding torrent file. This step is usually done over HTTP and completed in one connection, so this kind of traffic is negligible.

Downloaders periodically check in with the tracker to keep it informed of their progress and receive lists of peers, which operates over TCP. Many of these flows have the signature of "get/anounce?" in the payload. We call the direction from the connection initiator to the acceptor the *request direction*, and the reverse direction the *response direction*. For some of the flows, in addition to the three-way handshake in the establishment of a TCP connection and the four-way handshake in its termination, there is one packet in the request direction and one packet in the response direction, and the sizes of the request packets in different flows are similar. Some properties of these flows can be summarized in the following rule:

rule1: *TCP, the same source IP, 5 request packets, 4 response packets, fixed size in request bytes.* Rule1 means that there are some TCP connections that are sent from the same source IP address. Each connection has 5 request packets and 4 response packets, and the total number of bytes in these request packets is fixed. Other flows may have different properties. For example, the number of packets in a TCP connection termination is less than 4, so the numbers of request packets or response packets are different from the ones in rule1. The properties of these flows can be represented by another rule.The third kind of traffic is between downloaders. Downloaders upload and download files from each other via direct connections. In the original protocol, this traffic operates over TCP. Later in some extended versions, UDP is also used. Most of these flows have the payload signature "bittorrent protocol". An example of the flow properties of this kind of traffic is in rule2. Rule2 represents properties of those flows that are used to download files, so they have large flow duration, and large number of response packets and response bytes.

rule2: *TCP, the same source IP, the number of response packets > 100, the number of response bytes > 100,000, flow duration > 20 s.* The fourth kind of traffic is communication between DHT nodes. In trackerless DHT protocol [2], BitTorrent uses a "distributed sloppy hash table" (DHT) for storing peer contact information for "trackerless" torrents. In effect, each peer becomes a tracker. BitTorrent clients include a DHT node, which is used to contact other nodes in the DHT to get the location of peers from which to download. This protocol is implemented over UDP. Payload signatures "d1:ad2:id20:" and "d1:rd2:id20:" are present in this kind of traffic. The following are examples of flow properties:

rule3: *UDP, the same source IP, the same source port, 1 request packet, 1 response packet, fixed size in request bytes, fixed size in response bytes.*

rule4: *UDP, the same source IP, the same source port, 1 request packet, 0 response packet, fixed size in request bytes.*

rule5: *UDP, the same source IP, the same source port, 2 request packets, 2 response packets, fixed size in request bytes. Rule3 is one example for the "GET_PEERS" request in DHT protocol. All flows are from the same source IP address and the same source port number. There are one request packet and one response packet, and both have a fixed number of bytes in all flows. Rule4 is the case that there is no response to the "GET_PEERS" request, so the number of response packets is 0. Rule5 is an example of an "ANNOUNCE_PEER" message after the "GET_PEERS"*

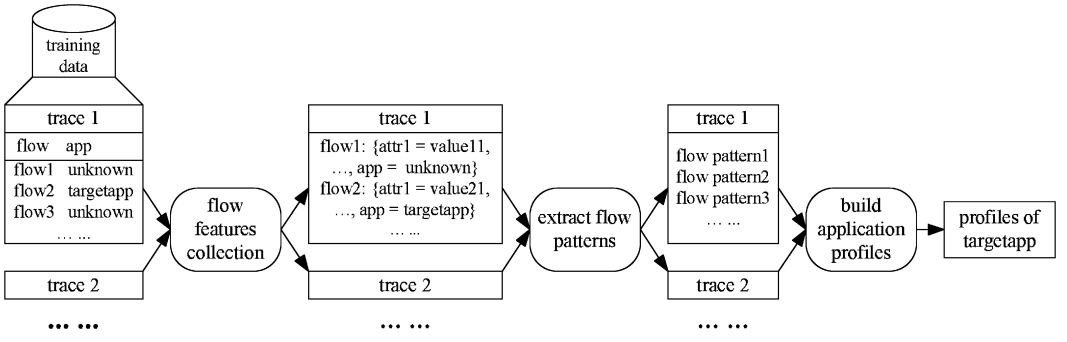

Fig. 1. Main steps of building application profiles.

*request, so there are two request packets and two response packets.*From the above example, we can see that P2P applications usually have several different types of traffic. There are even different cases in each type of traffic. We summarize the properties of the flows in each type of traffic in rules like rule1–rule5. Flow properties include the transport-layer information (such as protocol, whether all flows use the same IP address and/or the same port number) and statistical information (such as the number of packets, the number of bytes and flow duration, etc.). The profiles of an application are a set of such rules, which characterize the most important communication patterns of this application.

4. Application profiles

In this section, we explain how to build profiles for a given P2P application. Fig. 1 describes the main steps. To build application profiles, training traces are needed. In the training traces, we know in advance which traffic belongs to the target application and which one does not. After that, we determine what features should be chosen to construct profiles and how to collect these features. Having identified these flow features, flows in each training trace are represented in the form of $\{attribute_1 = value_1, \ldots, attribute_n = value_n\}$. We then extract flow patterns by discovering common properties shared by some flows of the application. Finally, we build application profiles based on the resulting flow patterns. We discuss each step in the remainder of this section.

4.1. Training traces

To build application profiles for a given P2P application, we use training traces that contain labeled traffic of the target application. Basically there are two methods to obtain this labeled training trace. One is to capture real traffic from the Internet and then identify and label the target application in the traffic trace. This step is time-consuming, and it is very difficult to get 100% accurate results. Another method is to get the training traces in a controlled environment, where one can accurately indicate which traffic belongs to the target application. We used real campus traffic traces for both training and evaluation. The datasets are packet traces collected in the gateway of our department. We captured the packet header and the first 42 bytes of payload of the traffic through the gateway. For privacy, we anonymized all the IP addresses. Based on the requirement of the network administrator, we also stripped the payloads of some well-known applications that could reveal the url, email address, and other user information.

We combined several methods to identify the target application, including payload-based methods, {IP, port} pairs, and even manual analysis. In the payload-based method, protocol signatures are identified either from previous studies [13,22] and public documents, or by reverse-engineering. In this step, we process packet traces with Bro [20], an open-source network intrusion detection system. We added functions to Bro to match the signatures of a given application in packet payloads and label the corresponding connection records. For example, the signature of BitTorrent is represented by this regular expression: "^get/announce\?| bittorrent protocol |d1:ad2:id20:|d1:rd2:id20: | ^azver".

Not all flows in the target application have traceable signatures. We use {IP, port} pairs and even manual analysis to assist identification. {IP, port} pairs associate a particular IP address and a specific port with one application, since the service reflected at a specific port for a specific IP does not change over a short period. For example, if connection $C1$ and $C2$ have the same source IP and the same source port and are open within the same time interval, then we consider that $C1$ and $C2$ are involved in the same application. Note that this method may have problems in some cases of network address translation (NAT), which do not appear in our datasets.

We denote the target application as *tapp*. We denote by *tappinst* an instance of *tapp* that is employed by a specific user at a specific time. To build application profiles for *tapp*, we first study the communication pattern of each individual *tappinst*, and then merge their patterns together. Therefore, each training trace that we choose consists of the flows of one *tappinst* and some other traffic (not *tapp*) as the background data. If there is more than one *tappinst* in the original traffic (e.g., several people in our department used BitTorrent to download files at the same time), we separate the original traffic into multiple training traces such that each trace contains one *tappinst*.

4.2. Flow features collection

To build profiles, the first question is to determine what features should be chosen to construct profiles and how to collect these features. As we have mentioned, application profiles are a set of rules and each rule summarizes the common properties shared by a number of flows in the application, which include transport-layer properties and statistical properties. We focus on properties of bidirectional flows, also known as connections. The reason for this is that we need to differentiate the statistical observations on the request and the response direction, which are different in many applications.

Typically, a flow is defined by the five-tuples {srcIP, destIP, srcPort, destPort, protocol}. We use the five-tuples to study transport-layer properties. In addition, flow statistics are also chosen as flow features to study statistical properties. Previous studies [19,23,25] have investigated the issue of feature selection in traffic classification. For instance, Moore et al. [19] applied Bayesian analysis techniques to categorize traffic by application. They capitalized on hand-classified network data, using it as input to a supervised Bayes estimator. Two hundred and forty eight flow features were considered in training the classifier. They used fast correlation-based filter (FCBF) for discriminator selection and dimension reduction. Based on previous studies and our own experiences, we chose several flow statistics, as summarized in Table 1.

With the exception of flow duration, the other flow statistics are calculated in the request and response direction separately. For example, "rpack" represents the number of

packets in the request direction and "ppack" represents the number of packets in the response direction. For the number of bytes (or packet size), we only calculate the payload size, since the sizes of IP and TCP/UDP headers are not important for application identification. Besides the flow statistics that are commonly used, we add four additional statistics to represent the size of the first (second) data packet in the request (response) direction, i.e., *rhbyte1*, *phbyte1*, *rhbyte2*, and *phbyte2*. The first data packet means the first packet after the three-way handshake in TCP connection establishment. For the UDP protocol, its first packet is also the first data packet. The first few packets in connections are important for application identification since they usually capture the application's negotiation stage.

The flow features that are chosen to construct application profiles are the basic five-tuples plus all the flow statistics summarized in Table 1. We further divide the flow statistics into *axis attributes* and *extra attributes*. Intuitively the axis attributes are the essential attributes of a connection, while the extra attributes provide complementary information. The five-tuples of flow are also regarded as axis attributes. The axis attributes are applicable to all connections, while the extra attributes are only used to describe large flows (i.e., where the number of packets is large), since the average packet size and the inter-arrival time between packets do not have much meaning when there are only a few packets in a connection. In addition, the axis and extra attributes are handled differently in the later steps in building application profiles.

To collect the flow features, we process packet traces with Bro. All the flow statistics can be updated on-line in a streaming fashion, which means that we do not need to store data per packet, but rather per connection.

4.3. Extract flow patterns

As we have mentioned, P2P applications usually have different types of traffic, and flows in each type of traffic share some common properties. We call these common properties *flow patterns*, as summarized in rule1–rule5. The profiles of an application are a set of such rules. Manually extracting flow patterns from large numbers of flows is hard and time-consuming. We use data mining techniques for the extraction of frequent itemsets and association rules [10] in this step. Association rule mining is used to find inter-relationships (correlations) among members of a data set. The application of association rule mining to network traffic is widely deployed in the realm of network security, especially intrusion detection [15,16], and recently in traffic analysis [4].

A set of items is referred to as an *itemset*. The occurrence *frequency* of an itemset is the number of transactions that contain the itemset, which is also known as the *support* of the itemset. If the support of itemset I satisfies a predefined minimum support threshold, then I is a frequent itemset. *Association rules* are extracted from frequent itemsets and show correlations among contained elements. Association rules are usually in this form: $A \Rightarrow B(support, confidence)$, where A and B are itemsets, and $A \cap B = \phi$. A is often referred to as the *body* of the rule, while B as the *head* of the rule. The support is the number of transactions that contain $A \cup B$, divided by the total number of transactions. The confidence is the support of sets that contain $A \cup B$, divided by the support of sets that contain A. Finally, we have:

$$support(A \Rightarrow B) = P(A \cup B) \tag{1}$$

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)} \tag{2}$$

Association rule mining can be viewed as a two-step process: (1) find all frequent itemsets that occur at least as frequently as a predetermined minimum support, (2) generate association rules from the frequent itemsets, which must satisfy minimum support and minimum confidence requirements. Apriori [3] is a widely used algorithm for mining frequent itemsets and association rules.

Association rule mining satisfies our need to extract common properties shared by some flows (flow patterns) from a large number of flows, since we need to discover correlations between flow patterns and the target application. Based on the flow features and the application labels, we develop various attributes for the flows. Each flow is represented in the form of $\{attribute_1 = value_1, \ldots, attribute_n = value_n\}$. The attributes include the flow features and the application label, and the values are the corresponding values of each attribute. The problem is that Apriori is an algorithm for mining frequent itemsets for

Table 1
The flow statistics.

Statistics	Meaning
Axis attributes	
rpack (ppack)[a]	Number of packets in request (response) direction
rbyte (pbyte)	Number of bytes
rhbyte1 (phbyte1)	Size of the first data packet
rhbyte2 (phbyte2)	Size of the second data packet
Extra attributes	
Duration	Flow duration
rsizeavg (psizeavg)	Average packet size
rsizevar (psizevar)	Variance of the packet size
rduravg (pduravg)	Average inter-arrival time between packets
rdurvar (pdurvar)	Variance of inter-arrival time between packets

[a] Statistics that begin with "r" represent the statistics in the request direction, and statistics that begin with "p" denote the ones in the response direction.

Boolean association rules. However, flows have richer attribute types. The five-tuples of flows are categorical, while the other flow statistics are quantitative. Boolean attributes can be considered as a special case of categorical attributes. Categorical attributes can be processed by Apriori using the following method. We input transactions in the form of {srcPort = 2119, destPort = 80, ...} to Apriori, then Apriori will regard "srcPort = 2119" as a Boolean attribute, instead of regarding "srcPort" as a categorical attribute.

Unlike categorical attributes, quantitative attributes cannot be simply processed as boolean attributes. Quantitative attributes have different types (some are in integer form, e.g., rpack and rbyte, and others are in floating-point form, e.g., rsizeavg), and have a wide range of values defining their domain. In addition, quantitative attributes have an implicit ordering among their values. For example, {rbyte = 150} and {rbyte = 151} are similar statistical observations for the attribute "rbyte". To do this, we first partition the ranges of those quantitative attributes into "bins". The partitioning strategy that we use is *equal-frequency binning* [10], where each bin has approximately the same number of tuples assigned to it. Another common strategy is *equal-width binning* where the interval size of each bin is the same. We choose equal-frequency binning rather than equal-width binning because of the uneven distribution of these quantitative attributes. With the partitions of the quantitative attributes, we then replace the exact values of the flow attributes with the ranges of the bins that contain the exact values. An example is given below.

Example 1. {srcIP = 193.169.140.183, srcPort = 3946/6, rpack = 11, rbyte = 408...450, duration = 4.0...22870.5, rsizeavg = 34.7...43.1, rhbyte1 = 156, rhbyte2 = 168, ..., signame = bittorrent}.

We regard the five-tuples of a flow as four keys: (srcIP, destIP, srcPort, destPort), because port numbers are meaningful only when combined with protocol type. In the example, "srcPort = 3946/6" means the port number is 3946 and the protocol type is 6 (TCP). Some bins only have one value because there are enough flows having this value for the corresponding attribute (e.g., rpack). As for the four flow features rhbyte1, rhbyte2, phbyte1, and phbyte2, they are intended to capture the application's negotiation stage. Therefore, their actual values are kept, and no partition is processed. "signame = bittorrent" means this connection is identified as "BitTorrent".

The only parameter that needs to be set in this step is the number of bins for each quantitative attribute. We choose a larger number of bins for the axis attributes (eg. 20 for rpack and rbyte), and a smaller number of bins for the extra attributes (eg. 5 for duration). For flow duration, we only care whether the flow is short or long, and the inter-arrival time between packets would be affected by network conditions.

To automatically extract common properties shared by partial flows of the target application from the large number of flows, we input all the flows in a training trace to Apriori [1] in the form as in Example 1. To improve the efficiency and reduce the number of association rules that are generated, we restrict the head of the rules to the target application, since we want to discover the correlations between flow patterns and the target application. Examples of the resulting association rules are given below.

Example 2. signame = bittorrent $\Leftarrow$ srcIP = 193.169.140.183, srcPort = 19270/17, rpack = 1, ppack = 1, rbyte = 98, pbyte = 272 (721, 100.0). signame = bittorrent $\Leftarrow$ srcIP = 193.169.140.183, srcPort = 19270/17, rpack = 1, ppack = 0, rbyte = 98 (1311, 100.0).

Example 2 illustrates the resulting association rules. The heads of all the rules are restricted to "signame = bittorrent" when the target application is "BitTorrent", and the body of the rules contains various flow properties. In the first rule, in the (support, confidence) pair "(721, 100.0)", the 721 means that there are a total of 721 connections satisfying this rule, and 100.0 means that the confidence of the rule is 100%.

4.4. Build application profiles

Given these flow patterns, the next step is to build profiles for the target application. Several issues need to be considered here. First, there are usually a large number of association rules generated from Apriori, so we want to reduce the number of rules by removing redundant rules and keeping meaningful ones. The second issue is how to get a common set of flow patterns across different training traces. As we know, network applications may have different versions of software implementation. In addition, different configurations or even different users may result in various flow patterns. Therefore, we need to combine the different flow patterns to get a common set. The third question is whether there is any correlation among these different flow patterns? We discuss ways to solve these problems in the remainder of this section.

4.4.1. Maximal association rules

A major challenge in association rule mining from a large data set is the fact that such mining often generates a huge number of frequent itemsets and association rules satisfying the minimum support (*min_sup*) and minimum confidence (*min_conf*) conditions, especially when these are set low. To overcome this difficulty, the concepts of *closed frequent itemset* and *maximal frequent itemset* are introduced [10]. An itemset X is a closed frequent itemset in a data set S if X is frequent, and there exists no proper super-itemset Y such that Y has the same support as X in S. An itemset X is a maximal frequent itemset in S if X is frequent, and there exists no proper super-itemset Y such that $X \subset Y$ and Y is frequent in S.

As shown in Example 2, the heads of the association rules are restricted to "$signame = tapp$", and the bodies of the rules contain various flow properties. Acting as flow patterns in application profiles, the bodies of the association rules had better contain more flow properties, so *maximal association rules* are kept. In other words, we keep the association rule $r1$ "$signame = tapp \Leftarrow B1$" if $r1$ satisfies *min_sup* and *min_conf*, and there exists no association rule $r2$ "$signame = tapp \Leftarrow B2$" such that $B1 \subset B2$ and $r2$ satisfies *min_sup* and *min_conf*. Here, $B1$ and $B2$ are itemsets

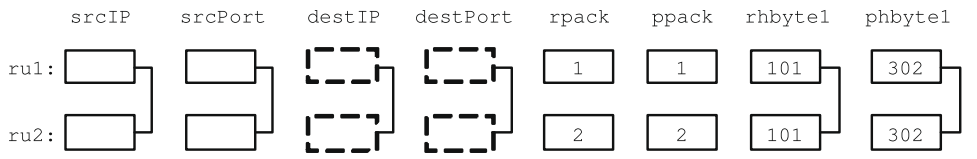

Fig. 2. Correlations among *ru1* and *ru2*.

with items in the form of "$attribute_i = value_i$". This step of the process can greatly reduce the number of association rules while keeping most of the information.

4.4.2. Generalize rules

Network applications usually have different versions or software implementations. In addition, different configurations or even different users may result in different flow patterns. To build profiles for the target application, we try to capture the most significant communication patterns of this application, which are common in each instance of the target application. So we need to merge different rules from different training traces together. Our approach of merging rules is based on the fact that even the same behavior will have differences between different training traces. Therefore, we should not expect the mined rules to match exactly. Instead we need to combine similar patterns into more generalized ones.

Two methods are used to generalize rules. The first one is to *normalize attribute values*. Take the following association rule in Example 2 as an example: *signame = bittorrent ⇐ srcIP = 193.169.140.183, srcPort = 19270/17, rpack = 1, ppack = 1, rbyte = 98, pbyte = 272 (0.40/721, 100.0)*. This rule indicates that 721 bittorrent flows have the same srcIP, srcPort and rpack, etc. From another training trace, we may get a similar rule but with different srcIP and srcPort. For IP address and port number, we only care if the flows are sent from (or received by) the same IP address and port number, but we do not care about the specific values. Therefore, we normalize these attribute values across different training traces.

Formally, suppose there are two rules $r3$ and $r4$,

$$r3 : signame = tapp \Leftarrow a_1 = vx_1, \ldots, a_j = vx_j, a_{j+1} = vx_{j+1}, \ldots, a_k = vx_k$$

$$r4 : signame = tapp \Leftarrow a_1 = vy_1, \ldots, a_j = vy_j, a_{j+1} = vy_{j+1}, \ldots, a_k = vy_k$$

where $\{a_i, 1 \leqslant i \leqslant k\}$ are the attributes, which are the same in $r3$ and $r4$, and $\{vx_i, 1 \leqslant i \leqslant k\}$ and $\{vy_i, 1 \leqslant i \leqslant k\}$ are the corresponding values. Attributes $\{a_i, 1 \leqslant i \leqslant j\}$ are from the four flow keys, while attributes $\{a_i, j+1 \leqslant i \leqslant k\}$ are from the other flow statistics. If $\{vx_i = vy_i, j+1 \leqslant i \leqslant k\}$, then $r3$ and $r4$ will be merged together. The merged rule is represented by

$$r5 : signame = tapp \Leftarrow a_1, \ldots, a_j, a_{j+1} = vx_{j+1}, \ldots, a_k = vx_k$$

We keep the attributes $\{a_i, 1 \leqslant i \leqslant j\}$ in $r3$, which shows that the flows have the same value for those attributes. In addition, if $\{vx_i = vy_i, 1 \leqslant i \leqslant j\}$, then the actual value for the attribute $\{a_i\}$ will also be kept in the merged rule. This means that the actual value is important for the application, especially when $\{a_i\}$ is srcPort or destPort. The following is an example of the sort of rule obtained by merging rules from different trace files.

Example 3. signame = bittorrent ⇐ srcIP, srcPort, rpack = 1, ppack = 1, rbyte = 98, pbyte = 272.

In addition to normalizing the attribute values, another method for generalizing rules is to *extract a common rule subset*. Assume we get a rule $r6$ "$signame = tapp \Leftarrow A1$" from one training trace, and another rule $r7$ "$signame = tapp \Leftarrow A2$" from another trace, then the merged rule becomes $r8$ "$signame = tapp \Leftarrow A1 \bigcap A2$", as long as $r8$ itself is a mined rule (i.e., satisfies *min_sup* and *min_conf*). To reduce the number of association rules and improve the interpretability of the model, we have already obtained maximal association rules for each training traces. However, some maximal association rules are not exactly the same across different traces, so their intersections are kept in the merged rules.

The basic association mining algorithms do not consider any domain knowledge and as a result they can generate some "irrelevant" rules. The issue of finding interesting rules from large sets of discovered association rules has been studied in [14]. Klemettinen et al. propose to use rule templates specifying the allowable attributes values to post-process the discovered rules. A challenge of this approach is that one has to know what rules are interesting, which demands strong prior knowledge on the target application. We use the axis attributes and extra attributes as item constraints. As we have mentioned in Section 4.2, the axis attributes are the essential attributes of a connection, while the extra attributes provide complementary information to large flows. As a form of item constraints for interesting rules, we require that the body of the association rule must contain at least three axis attributes.

4.4.3. Correlation among flow patterns

Until now, we have been discussing how to find interrelationships between items of one record (connection), where each relationship is represented by one association rule. In addition to these intra-record relationships, we further discover inter-record patterns, i.e., correlations between the flows that satisfy different rules. Suppose there are two rules, *ru1* and *ru2* as follows.

Example 4. ru1: signame = bittorrent ⇐ srcIP srcPort rpack = 1 ppack = 1 rhbyte1 = 101 phbyte1 = 302 (133). ru2: signame = bittorrent ⇐ srcIP srcPort rpack = 2 ppack = 2 rhbyte1 = 101 rhbyte2 = 149 phbyte1 = 302 phbyte2 = 53 (387).

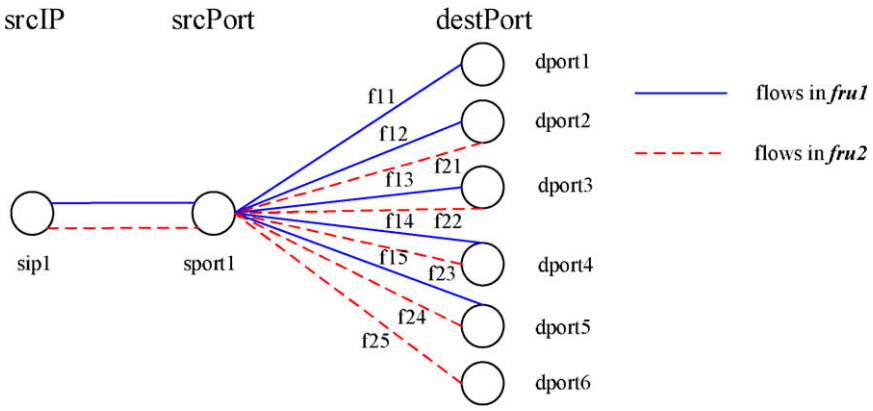

Fig. 3. The correlation on destPort.

In one instance of BitTorrent, there are 133 flows satisfying *ru1* and 387 flows satisfying *rul2*. We call the set of the 133 flows *fru1*, and the set of the 387 flows *frul2*. Among these flows, if no flow satisfies both *ru1* and *ru2* simultaneously, i.e., $frul1 \bigcap frul2 = \Phi$, then we call these two rules *mutex*.

Ru1, *ru2* and their correlations are depicted in Fig. 2. The first line of rectangles indicates the attributes and corresponding values of *ru1*, and the second line indicates *ru2*. Each rectangle represents one attribute whose label is above the rectangle, and the number inside the rectangle is the value for the attribute. For those rectangles denoted by dashed lines, the rule does not contain the attributes, but there are correlations between them. For brevity, some attributes (e.g., rhbyte2) are not shown in the figure. Correlations are represented by the lines between the rectangles.

The first correlation is that the values for srcPort in *ru1* and *ru2* are the same. In other words, not only all flows in *fru1* have the same srcPort1, and the flows in *fru2* have the same srcPort2, but also srcPort1 = srcPort2. The second and third correlations exist in the attributes of destIP and destPort, which even do not appear in *rul1* and *rul2*. Fig. 3 explains the correlation on destPort. In the figure, each node indicates one value for the three attributes: srcIP, srcPort and destPort, and each line indicates one flow. The solid lines represent the flows in *fru1* (e.g., f11, f12, f13, ...) while the dashed line represent the flows in *fru2* (e.g., f21, f22, f23, ...). All flows are sent from the same host and the same srcPort. The correlation in destPort indicates that although flows in *fru1* use different destPort and flows in *fru2* also use different destPort, some flows in *fru1* have the corresponding flows in *fru2*, which use the same destPort as the flows in *fru1*, such as f12 and f21, f13 and f22, and so on. The other three correlations, in srcIP, rhbyte1 and phbyte1, are more obvious and can be directly observed from the two rules.

Next we describe the practical meanings of *ru1*, *ru2* and their correlations. We have given an example of an application profile for BitTorrent in Section 3. **Rule3** to **rule5** present examples of flow patterns for the DHT traffic. In **Example 4**, *ru1* matches **rule3** and *ru2* matches **rule5**. All the flows in *fru1* and *fru2* are from one DHT node since they use the same srcIP. Most of the flows in *fru1* and *fru2* use different destIP, so this DHT node communicates with different hosts. The correlation in srcPort indicates that this DHT node uses the same srcPort for all the communication with different hosts, while the correlations in destPort and destIP indicate that there are some flows in *fru1* communicating with the same destIP and using the same destPort as that of some flows in *fru2*. The correlations among different flow patterns not only provide more information for the application profile, but can also be used in the application identification step, as will be discussed later.

4.4.4. Description of steps

After describing several approaches to processing association rules, we can now summarize the main steps involved in building application profiles:

(1) partition quantitative attributes;
(2) prepare training trace files (each for one instance of *tapp*): label connections of *tapp* and collect features for each connection;
(3) for each training trace file use Apriori for association mining; get maximal association rule;
(4) merge rules from multiple training traces;
(5) find correlations among flow patterns.

We use *tapp* to denote the target application. The first step is to partition the ranges of the quantitative attributes into intervals, as described in Section 4.3. We then choose several training trace files, each including flows of one instance of *tapp* and the corresponding background flows. For each file, we label all connections that are identified as *tapp* using the methods explained in Section 4.1, and then collect the features of each connection as described in Section 4.2, so that each connection is represented in the form of Example 1. Next, we input all the connections in a training trace file to Apriori for association rule mining. We use all the connections for mining, instead of only the connections belonging to *tapp* since we need some background data. If we only used the connections belonging to *tapp*, we could not be sure that the resulting association rules were unique to *tapp*. In other words, other applications may also have the patterns described by the association rules. After that, we restrict the mined rules to

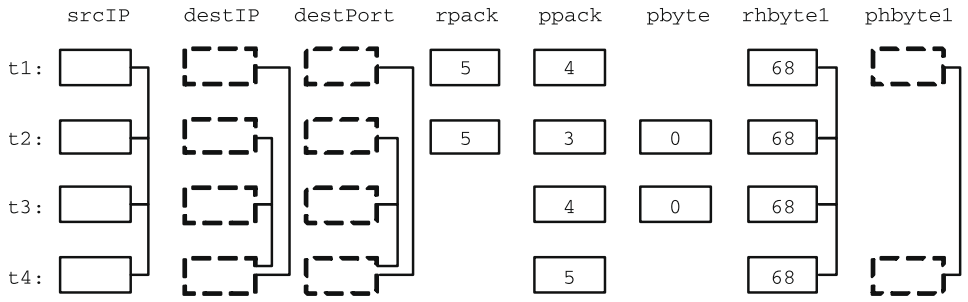

Fig. 4. Examples of correlations among flow patterns.

maximal association rules to reduce the number of association rules, and merge the rules from multiple training traces to obtain generalized rules. The last step is to find correlations among the resulting flow patterns.

TCP and UDP connections are mined and processed separately. The resulting TCP and UDP flow patterns are represented as *gTcprul* and *gUdprul*, respectively. Next we give an example of the resulting profile for BitTorrent. We select five training traces for the target application from the campus dataset described in Section 4.1. Each trace contains one instance of BitTorrent. The training traces are processed by the steps discussed in the previous part. Finally we get 15 rules for *gTcprul* and 12 rules for *gUdprul*. Some of them are given in Table 2. For brevity, the head of the rules is shown as "bittorrent" to represent "signame = bittorrent".

From these examples, one can see that the flow patterns are coincident with the application profile examples given in Section 3. Take **t1** as an example: it depicts the behavior patterns of some connections from the same srcIP. Each connection has five packets in the request direction and four packets in the response direction. The size of the first data packet (and the only data packet) in the request packet is 68. This flow pattern matches ***rule1*** very well, which describes the traffic by which downloaders periodically talk to the tracker. **t2** also agrees with ***rule1***. In this case, the downloader does not get any response from the tracker (pbyte = 0). **t5** agrees with ***rule2***, which is used for downloading files between peers. **u1**–**u6** are flow patterns for UDP, following ***rule3*** to ***rule5***, respectively. These flow patterns describe communication patterns between DHT nodes.

Fig. 4 gives the correlations among **t1**, **t2**, **t3** and **t4**. Among the four flow patterns, **t1** and **t3** are not mutex, and we only find correlations among mutex rules. Compared with Fig. 2, a new kind of correlation appears on phbyte1. This correlation indicate that although not all flows satisfying **t1** have the same phbyte1, some flows satisfying **t1** have the same phbyte1 as other flows satisfying **t4**. Fig. 4 only gives some examples of the correlations, and there are similar correlations among other flow patterns. The resulting application profiles are used in the application identification step. The flow patterns can also be directly reported to the network operators, because of their good readability.

5. Application identification

Based on the behavioral profiles of network applications, we use a two-level matching method to identify new traffic. Before describing the two-level matching method, we first discuss a simple alternative method. The simple method is to observe the properties of each individual connection by comparing this connection with each rule in the profile. If the connection satisfies any of the rules, then it is identified as the target application, since every rule is considered as a flow pattern. There are two problems with this simple method. First, it will have

Table 2
Examples of flow patterns for BitTorrent.

Flow patterns for TCP
t1: bittorrent ⇐ srcIP rpack = 5 ppack = 4 rbyte = 53...69 rhbyte1 = 68
t2: bittorrent ⇐ srcIP rpack = 5 ppack = 3 rbyte = 53...69 pbyte = 0 rhbyte1 = 68
t3: bittorrent ⇐ srcIP ppack = 4 rbyte = 53...69 pbyte = 0 rhbyte1 = 68
t4: bittorrent ⇐ srcIP ppack = 5 rhbyte1 = 68
t5: bittorrent ⇐ srcIP rpack = 64...746 rbyte > 13,185 rhbyte1 = 68 rsizeavg > 416.4
t6: bittorrent ⇐ srcIP rhbyte1 = 68 phbyte1 = 68 duration > 25.3

Flow patterns for UDP
u1: bittorrent ⇐ srcIP srcPort rpack = 1 ppack = 1 rhbyte1 = 101 phbyte1 = 302
u2: bittorrent ⇐ srcIP srcPort rpack = 1 ppack = 0 rhbyte1 = 101
u3: bittorrent ⇐ srcIP srcPort rpack = 1 ppack = 1 rhbyte1 = 98 phbyte1 = 272
u4: bittorrent ⇐ srcIP srcPort destPort rpack = 1
u5: bittorrent ⇐ srcIP srcPort rpack = 2 ppack = 2 rhbyte1 = 101 phbyte2 = 62
u6: bittorrent ⇐ srcIP srcPort rpack = 2 ppack = 2 rhbyte1 = 101 rhbyte2 = 149 phbyte1 = 302 phbyte2 = 53

a high false positive rate, since some of the flow patterns are not unique to the target application. Second, the application profile is set up to discover the most significant flow patterns of the target application. Some flows that belong to *tapp* are not described by these flow patterns, so there will be a large false negative rate. We use a two-level matching approach to solve these two problems.

5.1. Host-level matching

To identify *tapp* in a new trace file, we first locate those hosts that participate in *tapp*. In studying the behavior of each host, the first step is to group all connections sent from and received by this host together. To avoid double counting the connections, we assign each connection to only one side of the communication (either the source or the destination). The connection is assigned to the host that has the larger number of connections, since we can get a more complete view of its behavior. For example, for the P2P applications in our trace files, we focus on the behavior of the hosts inside our department instead of those outside. The reason for this connection assignment is that we can get a complete view of the behavior of the hosts inside the department, but only a partial view of those outside. Next, for each host, we find the flow patterns Rt_i in *gTcprul* that are satisfied by connections of this host. To compare connections with flow patterns, we need to examine multiple connections simultaneously, taking the normalized attributes into consideration.

u1: *bittorrent* ⇐ *srcIP srcPort rpack = 1 ppack = 1 rhbyte1 = 101 phbyte1 = 302*.

For example, we compare a host's connections with **u1**. This rule is mined from multiple connections, i.e., it satisfies a minimum support (*min_sup*). Therefore, it is necessary that multiple connections satisfy **u1**, and these connections should have the same srcIP and srcPort, although their exact values are not restrained.

We denote the number of connections that satisfy Rt_i by $NumConn(Rt_i)$, and the total number of Rt_i that satisfied by connections as $NumRul(Rt)$. Similarly, we also calculate $NumConn(Ru_i)$ and $NumRul(Ru)$ for UDP. If $NumRul(Rt)$ and $NumRul(Ru)$ are large, which means this host satisfies multiple-flow patterns of the target application, and if $NumConn(Rt_i)$ and $NumConn(Ru_i)$ are large, which means there are many connections of the host that match the flow patterns, then we conclude that this host participates in the target application (denoted as an *ahost*). This host-level matching that is based on *multiple-pattern* and *multiple-flow* can decrease the false positive rate, especially for complex network applications. If the flow patterns from the training trace files are complete and generalized, we hope this level of matching can identify the hosts participating in the target application with high accuracy and a low false positive rate.

Table 3
Examples of flow patterns for PPLive.

Flow patterns for TCP
t1: pplive ⇐ srcIP destPort rbyte = 81 pbyte = 0 rhbyte1 = 4 rhbyte2 = 61
t2: pplive ⇐ srcIP rpack = 6 ppack = 5 rbyte = 53…69 pbyte = 16 rhbyte1 = 4 rhbyte2 = 61 phbyte1 = 4 phbyte2 = 12
t3: pplive ⇐ srcIP rpack = 8 rhbyte1 = 4 rhbyte2 = 61 phbyte1 = 4
t4: pplive ⇐ srcIP ppack = 4 rhbyte1 = 4
t5: pplive ⇐ srcIP rpack = 1 ppack = 1 rbyte = 0…53 pbyte = 0 rhbyte1 = 28
Flow patterns for UDP
u1: pplive ⇐ srcIP srcPort rpack = 2 ppack = 2 rhbyte1 = 61 phbyte1 = 545 phbyte2 = 61
u2: pplive ⇐ srcIP srcPort rpack = 1 ppack = 1 rhbyte1 = 61 phbyte1 = 545
u3: pplive ⇐ srcIP srcPort rpack = 1 ppack = 1 rhbyte1 = 61 phbyte1 = 529
u4: pplive ⇐ srcIP srcPort rpack = 1 ppack = 0 rhbyte1 = 61
u5: pplive ⇐ destIP destPort rhbyte1 = 61

5.2. Flow-level matching

After we identify those *ahosts*, the second step is flow-level matching. Since a host may take part in different applications at the same time, in this step, we study all the flows of every *ahost* to determine which flows belong to this application and which ones do not. As we have mentioned, some flows that belong to *tapp* are not described by the flow patterns in the profile, since the application profile represents only the most significant flow patterns of the target application. There are two reasons for these false negatives.

The first one is that the flow patterns are maximal association rules since they contain more flow properties. However, the choice of maximal association rule is a lossy compression method. Some flows only satisfy a subset of the maximal rules. These flows cannot be identified if only the maximal rules are used in identification. To solve this problem, we use the *closed association rule* in flow-level matching, which is a lossless compression method. The rule $r1$ "$signame = tapp \Leftarrow B1$" is a closed association rule, if $r1$ satisfies *min_sup* and *min_conf*, and there is no association rule $r2$ "$signame = tapp \Leftarrow B2$" such that $B1 \subset B2$ and $r2$ has the same support as $r1$. For example, given the following two rules:

Example 5. r1: signame = bittorrent ⇐ srcIP srcPort rpack = 2 ppack = 2 rhbyte1 = 101 rhbyte2 = 149 phbyte1 = 302 phbyte2 = 53 (7792). r2: signame = bittorrent ⇐ srcIP srcPort rpack = 2 rhbyte1 = 101 rhbyte2 = 149 phbyte1 = 302 (8531).

There are 8531 flows satisfying *r2*. Among them, 7792 flows satisfy *r1*. In building the application profiles, we use the maximal association rule to get more flow properties in the flow patterns, so only *r1* is kept. However, if we use *r1* in flow-level matching, the other $(8531 - 7792)$ flows will be missed since they do not satisfy *r1*. Therefore, the closed association rule *r2* is used in this step.

The second reason is that to get a compact set of flow patterns, we set a relatively high *min_sup*. However, some patterns of *tapp* cannot be discovered since fewer than *min_sup* connections satisfy them. To identify these flows, we decrease *min_sup* and *min_conf* to get more rules. The new problem is that a low *min_sup* and *min_conf* may generate too many association rules, and some of them will be irrelevant. We use the correlations among rules to solve

this problem. Since *gTcprul* and *gUdprul* describe significant patterns of *tapp*, we only keep those new generated rules that have correlations with rules in *gTcprul* or *gUdprul*. This process can help to remove some irrelevant rules, which produce false positives in flow-level matching.

As shown in Figs. 2 and 4, we look for correlations on the following attributes: srcPort, destIP, destPort, rhbyte1 and phbyte2. Take destIP as an example: if some of the new generated rules have flow correlation on destIP with some rules in *gTcprul* or *gUdprul*, then these new rules are kept. The reason is that the flows with the same remote host are likely to belong to the same application. Finally, for each connection in *ahost*, if it satisfies any flow pattern in application profiles or the newly generated relevant rule, we classify this connection as belonging to the target application.

6. Experimental evaluation

We evaluate our approach using the campus traffic traces. In this section, we first describe the training trace files and validation trace files. After that, we show the training results and the accuracy of the application identification on validation trace files.

6.1. Experimental setup

The datasets that we use to evaluate our approach are packet traces collected in our department. We captured the packet header and the first 42 bytes of payload of the traffic going through the gateway. For privacy considerations, we anonymized the IP addresses and stripped the payloads of some well-known applications such as HTTP, FTP. We captured the traffic from October 16, 2006 to November 9, 2006, and from September 13, 2007 to October 2, 2007. We also generated packet traces of the two applications using several computers outside our department. This self-generated traffic can be accurately labeled.

For the reasons described below, we choose two popular P2P applications from the datasets to evaluate our approach: one is BitTorrent (P2P file-sharing), and the other is PPLive (P2P streaming). First, to build application profiles at the training phase, sufficient application instances are required from the trace data. There is enough traffic in our datasets for these two P2P applications. In addition, the prior labelling application on the training data must be accurate to guarantee the effectiveness of our approach. Although other P2P applications such as skype also appear in the datasets, it is very difficult to identify skype traffic from the training traces since it uses strong encryption mechanisms. In our experiments, both the training and the matching are performed offline. For the training stage, we chose trace files from the first period (October 16, 2006 to November 9, 2006). For the validation stage, we chose three kinds of trace files. One is other trace files from the first period, which represent the best sample. Another is trace files from the second period (September 13, 2007 to October 2, 2007), which we used to test the temporal stability. The last kind of file is the self-generated traffic traces.

6.2. Training results

We have given an example of the resulting profile for BitTorrent in Section 4.4. We selected five training traces. Each trace contains one instance of BitTorrent, and the duration for each instance is about several hours. There are 2000–5000 TCP BitTorrent connections, and 5000–50,000 UDP BitTorrent connections in each instance. We chose at most 50,000 UDP *tapp* connections, although some of the instances lasted longer than our time period, i.e., longer than 10 h. In building the application profiles, we needed to provide *min_sup* and *min_conf* when using Apriori for association mining. In this step, our method applies association mining to get significant flow patterns. To get a relatively compact set of flow patterns, we chose a relative high *min_sup*, which we set to 100–300 according to the number of *tapp* connections in each instance, and we set *min_conf* to 80%. The training traces are processed by the steps discussed in the previous sections. Finally, we obtained 15 rules for *gTcprul* and 12 rules for *gUdprul*. Some of these are given in Table 2.

We also tested the stability of the training results. We selected 4–8 training traces. The numbers of rules for *gTcprul* that we obtained varied between 14 and 19, and the numbers of rules for *gUdprul* range varied between 10 and 12. UDP flow patterns are very stable, since they are very similar across different instances. In addition, there are many UDP connections in each instance, which provide a large training dataset. Most of the rules from different sets of training traces are exactly the same, and at most one or two rules are different. The TCP flow patterns are relatively less stable than those for UDP, since the communication patterns for TCP connections are more complicated and there are less TCP connections in each instance. In the application identification step, we used the profiles obtained from five training traces.

Next we briefly introduce the training results for PPLive. From five training traces, we obtained 15 rules for *gTcprul* and 25 rules for *gUdprul*. Some examples of the flow patterns are given in Table 3.

6.3. Matching results

In this section, we present the accuracy results of identifying BitTorrent and PPLive in new trace files. We first present the best case result, that is, other trace files from the same venue and the same time period. The five training traces were chosen from between October 16 and October 30. In the validation stage, we chose 10 trace files from between October 16 and October 30 as well as between October 31 and November 9. Each validation trace contains about 50,000 connections. For BitTorrent, there are 1 or 2 instances in each validation trace. For PPLive, there are 1–3 instances in each trace.

In host-level matching, we identify the hosts that participate in the two applications with 100% accuracy and 0 false positives. In this step, we need to set thresholds for $NumRul(Rt)$ and $NumRul(Ru)$. Since the dominant communication patterns are described by application profiles, the thresholds can be safely selected from a range of values. For example, there are 15 rules in *gTcprul* and 12 rules

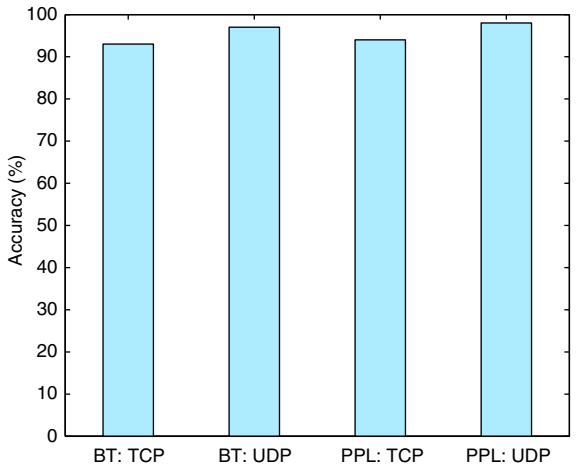

Fig. 5. Accuracy results.

in *gUdprul* for BitTorrent. For the hosts that have BitTorrent traffic in the validation traces, they satisfy 13–14 TCP flow patterns and 10–12 UDP flow patterns. On the other hand, other hosts satisfy at most 1–3 TCP flow patterns and 1 UDP flow pattern (**u4** in Table 2).

In the flow-level matching, we extended the rule set using the two methods described in Section 5.2. We obtained 39 TCP rules and 17 UDP rules for BitTorrent and 62 TCP rules and 70 UDP rules for PPLive. These extended rule sets were used for the flow-level matching. For the i^{th} trace file, we define app_i as the number of connections that participate in *tapp*, and $idencon_i$ as the number of connections that we identify as *tapp* and are also *tapp*. Then the average accuracy is defined by:

$$avgAccuracy = \sum_{i=1}^{N} idencon_i / \sum_{i=1}^{N} app_i$$

where N is the number of validation trace files.

The accuracy results for connections are given in Fig. 5. The four bars are *avgAccuracy* for TCP BitTorrent connections, UDP BitTorrent connections, TCP PPLive connections, and UDP PPLive connections, respectively. The results show that our approach has very high accuracy. Fig. 6 shows the percentage of the matched connections when we only use the flow patterns in the application profile, i.e., rules in *gTcprul* and *gUdprul* to do flow-level matching. The flow patterns in the profile are a compact set, and are used to describe significant communication patterns. In Fig. 6, we see that most of the UDP connections can be described by the dominant patterns. However, more than half of the TCP connections need to be identified by the extended flow pattern set generated in the flow-level matching step.

In addition to these 10 validation traces from the same period as the training traces, we also chose several validation traces from the second period, which was several months after the training traces, and several self-generated traces. In the second period, there is little BitTorrent traffic, because at that time, the P2P file-sharing traffic was required to go through a proxy server and was thus not included in the dataset. Therefore we can only validate PPLive in these traces. The evaluation results are shown in Table 4. "PPL2" means the PPLive traffic in the second period, "BTself" is the self-generated BitTorrent traffic, and "PPLself" is the self-generated PPLive traffic. The "app" column indicates the total number of connections in *tapp* in the selected training traces. "conn" means the total number of connections among all *ahost* (having *tapp* traffic), not the total number of connections in the trace files. "fpos" means false positives, which is the number of connections that we identified as *tapp* but are not *tapp*. From these results, one can see that our approach can obtain very high accuracy and very few false positives.

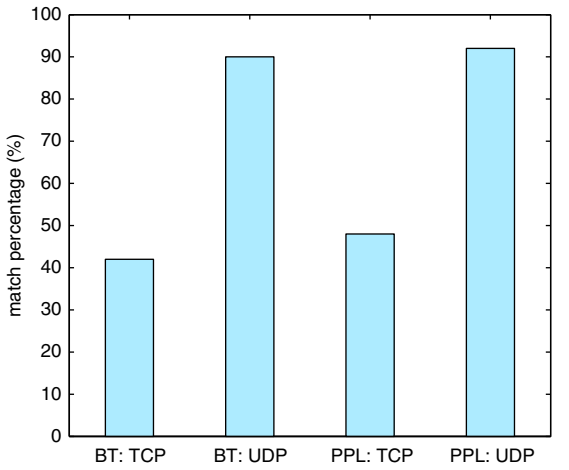

Fig. 6. Percentages of the matched connections.

Table 4
The accuracy results.

	conn	app	idencon	avgAccuracy	fpos
PPL2 (tcp)	14,475	9229	9058	98.1	206
PPL2 (udp)	129,627	127,757	123,618	96.8	712
BTself (tcp)	13,928	11,729	11,008	93.9	366
BTself (udp)	61,191	61,141	58,883	96.3	2
PPLself (tcp)	9239	8964	8301	92.6	52
PPLself (udp)	34,181	34,039	33,612	98.7	0

7. Discussion

7.1. Open issues

Our approach has several limitations. Note that many of the limitations are not specific to our approach, but are inherent to the problem. In this section, we discuss several open issues in traffic classification.

7.1.1. Evasion

The biggest challenge to traffic classification techniques in general is evasion. For example, *rhbyte1* (the first data packet in the request direction) is an important flow feature for building profiles and appears often in the resulting application profiles (e.g., Table 2). If the application developers

Table 5
Running time.

Process	Input	Output	Time
Training stage			
Prepare training trace	Traffic (18 h)	Training traces	52 min
	1,317,041 conn 272,280 tapp_conn	1,004,014 conn 267,000 tapp_conn	
Build profiles	Training traces	Application profiles	129 s
Validation stage			
Identify application	file1 (50,000 conn)		44 s
Identify application	file2 (100,000 conn)	Flows belong to *tapp*	188 s
Identify application	file3 (150,000 conn)		314 s

want to evade our method, they can modify the size of the first few packets in connections by randomly padding packet payloads. The accuracy of our approach would be decreased if those *rhbyte1* related properties were removed from the application profiles. However, all classification methods can be evaded: payload-based analysis cannot classify encrypted packets, port-based methods are defeated by a simple change of port, and approaches relying on flow statistics are sensitive to alterations of packet sizes and inter-arrival times.

Compared with other techniques, our approach should be more difficult to evade. This is because the application profiles summarize multiple aspects of communication patterns. First, the flow tuples describes the transport-layer properties, i.e., whether many flows use the same srcPort, or destPort or to the same destIP. Second, protocol signatures such as *rhbyte1* usually capture the application's negotiation stage. Third, flow statistics such as average packet size characterize traffic statistical properties. It is very hard for applications to disguise their behavior without adding large amounts of overhead. Nevertheless, building classifiers that are robust to evasion is an interesting topic for future research.

7.1.2. Training

Training is an important step for most traffic classification approaches using machine learning techniques. It has several challenges. First, the accuracy of the classifier highly depends on the quality of the training data. However, preparing accurately labeled and complete training data is very hard and time-consuming. If real traffic from the Internet is used, it is very difficult to accurately identify and label some applications, e.g., skype which uses encryption mechanisms. Some studies [6] propose possible approaches to identify Skype traffic, which can help to label skype traffic in the training data. We did not include skype in our experiments because these approaches cannot guarantee a very high identification accuracy. On the other hand, if the training data is generated in a controlled environment, then the completeness of the training data is hard to guarantee, although labeling traffic becomes easy. Second, since the classifier is learned from the training data, the accuracy of these approaches will decrease when new applications originate or old applications change their behaviors. In this case, we would like the classification system to adapt accordingly, i.e., automatically detect the need for retraining and update the classifier.

7.1.3. Encryption

Our approach is based on the packet header and flow statistics (packet size and inter-arrival time). Therefore, our technique can characterize encrypted traffic as long as the encryption is limited to the transport-layer payload. If layer-3 packet headers are also encrypted, such as in IPSec, then our approach cannot work. However, this is probably true for most classification techniques. Please note that this encryption issue is different from the one when we discussed skype. Our approach would be able to work on skype if we had adequate training data that accurately labeled skype traffic.

7.2. Practical considerations

7.2.1. Application identification vs. traffic classification

Our approach aims to identify a target application, especially P2P application. There are some studies focusing on the problems of classifying traffic flows into several predefined classes. These two problems are interrelated but are different in practice. One difference is that traffic classification methods usually cannot identify specific application sub-types [13]. For instance, they can identify P2P flows, but do not differentiate the specific P2P protocol (e.g., BitTorrent vs. Gnutella). Our approach can identify the specific P2P protocol or even different versions of one P2P protocol if they have marked difference. Another difference exists in preparing the training data. For application identification, we only need to label those connections that participate in the application, and do not need to determine the applications of other connections. However, in traffic classification, all connections should be labelled as one target class, which is more difficult.

7.2.2. Point of observation

In this paper, our approach is evaluated in traces collected at the edge of the network. In addition, almost all traffic from the targeted hosts is monitored. A complete view of the hosts' behavior facilitates both building good application profiles and accurately identifying the application in new traffic. The identification accuracy should not be affected if part of the targeted hosts' flows are missed. However, applying our approach at the backbone network may present different challenges. For example, individual user behavior might be hard to identify if only several of its flows are monitored.

7.2.3. Computational performance

In our experiments, both the training and the validation are performed offline. Table 5 gives some examples of the running time. The processing took place on a Dell optiplex GX755 with a 2 GHz processor and 2GB of memory. In the training stage, we first generated training traces from raw traffic, which last about 18 h. "1317041 conn" and "272280 tapp_conn" in Table 5 represent that there are

totally 1,317,041 connections and 272,280 connections of the target application. This step took about 52 min. Building application profiles from the training traces, including association mining using Apriori and the subsequent steps, was very fast. Therefore, the total running time of the training stage is less than 1 h. In Table 5, we also give the running times of identifying the target application in new validation files of different sizes. In the validation stage, the maximum memory usage is about 14%.

7.3. Future work

There are several interesting future directions. The first one is to extend our approach to build application profiles without training data, or with only part of the training data. With this improvement quite a few drawbacks of the current solution could be overcome. For example, it will be robust to the deliberate evasion issue. Identification for unknown application also would become possible.

The second one is to build general profiles from the traffic. Sometimes network administrators have specific requirements in identifying certain applications, such as identifying P2P traffic or anomaly detection. Other times, there are some other network management requirements which seek to have a more general understanding of the network traffic, such as what are the significant behaviors among the vast amount of traffic data. While there exist many prior works on anomaly detection and application identification, there has been less attempt to build general profiles from the traffic. Specifically, there are two aspects: (1) Automatically extract significant behaviors from the vast amount of unstructured traffic data, and (2) Provide profiles of these significant behaviors to help network operators understand the traffic in their network.

Limited by our trace files, we only evaluate two P2P applications with on-campus traffic traces. Another interesting study is to generate more traffic in a controlled environment. Take BitTorrent as an example, it has different versions of implementations. Training and validation traffic can be generated using multiple BitTorrent clients of different versions, tweaking all possible protocol's parameters (max bandwidth, number of connections, etc.) and even using some traffic shaping tools. By tuning training traffic, we can study which factors (different software versions, protocol's parameters, etc.) affect the resulting application profiles, and which one is irrelevant to the profiles. By tuning validation traffic, we can see how the approach performs with non-standard traffic profiles.

8. Conclusions

In this paper, we propose a novel profile-based approach to identify P2P applications. In contrast to classifying traffic based on statistics of individual flows, as has been done in previous studies, we build behavior profiles of the target application, which describe its significant patterns. We choose the five flow tuples and some flow statistics as flow features, and use association mining to acquire the correlations between various flow properties and the target application. To achieve a compact set of flow patterns and generalized rules, we obtain maximal association rules and merge rules from multiple training traces by normalizing the axis attributes. Correlations are then discovered among multiple-flow patterns. Using the resulting application profiles, a two-level matching method is used to identify the application in new traffic. We choose BitTorrent and PPLive to evaluate our approach with on-campus traffic traces. The application profiles generated are consistent with the application behaviors. The results show that our approach can obtain very high accuracy and very low false positive rates when identifying applications in validation trace files.

Acknowledgement

This work is funded by Hong Kong government's support via grant NSFC-RGC N_CUHK414/06.

References

[1] Apriori software. <http://fuzzy.cs.uni-magdeburg.de/~borgelt/apriori.html>.
[2] Bittorrent. <http://www.bittorrent.org/>.
[3] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proceedings of the 20th VLDB Conference, 1994.
[4] M. Baldi, E. Baralis, F. Risso, Data mining techniques for effective and scalable traffic analysis, in: Proceedings of Integrated Network Management, 2005.
[5] L. Bernaille, R. Teixeira, K. Salamatian, Early application identification, in: Proceedings of CONEXT, 2006.
[6] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, P. Tofanelli, Revealing skype traffic: when randomness plays with you, in: Procedings of SIGCOMM, 2007.
[7] M.P. Collins, M.K. Reiter, Finding peer-to-peer file-sharing using coarse network behaviors, in: Procedings of ESORICS, 2006.
[8] M. Crotti, F. Gringoli, P. Pelosato, L. Salgarelli, A statistical approach to ip-level classification of network traffic, in: Proceedings of ICC, 2006.
[9] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, C. Williamson, Offline/realtime traffic classification using semi-supervised learning, in: Performance Evaluation, vol. 64, 2007, pp. 1194–1213.
[10] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, 2001.
[11] F. Hernandez-Campos, F.D. Smith, K. Jeffay, A.B. Nobel, Statistical clustering of internet communication patterns, in: Computing Science and Statistis, vol. 35, 2003.
[12] T. Karagiannis, A. Broido, M. Faloutsos, K. Claffy, Transport layer identification of p2p traffic, in: Procedings of IMC, 2004.
[13] T. Karagiannis, K. Papagiannaki, M. Faloutsos, BLINC: multilevel traffic classification in the dark, in: Proceedings of SIGCOMM, 2005.
[14] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, A.I. Verkamo, Finding interesting rules from large sets of discovered association rules, in: Proceedings of CIKM, 1994.
[15] W. Lee, S. Stolfo, Data mining approaches for intrusion detection, in: Proceedings of the 7th USENIX Security Symposium, 1998.
[16] W. Lee, S.J. Stolfo, K.W. Mok, A data mining framework for building intrusion detection models, in: Proceedings of the IEEE Symposium on Security and Privacy, 1999.
[17] J. Ma, K. Levchenko, C. Kreibich, S. Savage, G.M. Voelker, Unexpected means of protocol inference, in: Procedings of IMC, 2006.
[18] A. McGregor, M. Hall, P. Lorier, J. Brunskill, Flow clustering using machine learning techniques, in: Proceedings of PAM, 2004.
[19] A.W. Moore, D. Zuev, Internet traffic classification using bayesian analysis techniques, in: Proceedings of Sigmetrics, 2005.
[20] V. Paxson, Bro: a system for detecting network intruders in real-time, in: 7th USENIX Security Symposium, 1998.
[21] 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 Internet Measurement Conference, 2004.
[22] S. Sen, O. Spatscheck, D. Wang, Accurate, scalable in-network identification of p2p traffic, in: Proceedings of WWW, 2004.
[23] N. Williams, S. Zander, G. Armitage, A preliminary performance comparison of five machine learning algorithms for practical ip

traffic flow classification, ACM SIGCOMM Computer Communication Review 36 (5) (2006).
[24] K. Xu, Z.L. Zhang, S. Bhattacharyya, Profiling internet backbone traffic: behavior models and applications, in: Proceedings of SIGCOMM, 2005.
[25] S. Zander, T. Nguyen, G. Armitage, Automated traffic classification and application identification using machine learning, in: Proceedings of LCN, 2005.

Yan Hu received her B.E. degree in Electronic Engineering from University of Science and Technology of China and M.E. degree in Communication and Information System from Chinese Academy of Sciences, PR China, in 2000 and 2003, respectively. She is currently a PhD student in the Department of Information Engineering at the Chinese University of Hong Kong. Her research interests include traffic measurement and analysis, network security.

Dah-Ming Chiu received a first degree from Imperial College, London, and a PhD degree from Harvard University. He worked in Bell-Labs, DEC and SUN before joining the Chinese University of Hong Kong in 2002. He is currently serving as the associate director of the university's Institute of Theoretical Computer Science and Communications (ITCSC); and an associate editor of IEEE/ACM Transactions on Networking (ToN). His current research interests include network resource allocation, routing, traffic measurement and analysis, P2P networking, wireless networks and economic issues in network architecture and operations.

John C. S. Lui received his PhD in Computer Science from UCLA. Currently, he is the chair of the Computer Science & Engineering Department at CUHK. His research interests span both in systems as well as in theory/mathematics with the emphasis on the robustness, scalability, and security issues on the Internet. John received various departmental teaching awards and the CUHK Vice-Chancellor's Exemplary Teaching Award, as well as the co-recipient of the Best Student Paper Awards in the IFIP WG 7.3 Performance 2005 and the IEEE/IFIP Network Operations and Management (NOMS) Conference. He is an associate editor of IEEE/ACM Transactions on Networking, IEEE Transactions on Computers, IEEE Transactions on Parallel and Distributed Systems, and Journal of Performance Evaluation.