Computer Communications 33 (2010) 1141–1150

Contents lists available at ScienceDirect

Computer Communications

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

Identify P2P traffic by inspecting data transfer behavior

Ke Xu [a], Ming Zhang [a,*], Mingjiang Ye [a], Dah Ming Chiu [b], Jianping Wu [a]

[a] Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science, Tsinghua University, Beijing 100084, PR China
[b] Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong, PR China

ARTICLE INFO

Article history:
Available online 18 January 2010

Keywords:
Traffic management
P2P traffic identification
Data transfer behavior
Content-based partitioning
Rabin fingerprint

ABSTRACT

Classifying network traffic according to its applications is important to a broad range of network areas. Since new applications, especially P2P applications, no longer use well-known fixed port numbers, the native port-based traffic classification technique has become much less effective. In this paper, we propose a novel approach to identify P2P traffic by leveraging the data transfer behavior of P2P applications. The behavior investigated in the paper is that downloaded data from a P2P host will be uploaded to other hosts later. To find the shared data of downloading flows and uploading flows online, the content-based partitioning scheme is proposed to partition the flows into data blocks. Flows sharing the same data blocks are identified as P2P flows. Theoretical analysis proves that the content-based partitioning scheme is stable and effective. Experiments on various P2P applications demonstrate that the method is generic and can be applied to most P2P applications. Experimental results show that the algorithm can identify P2P applications accurately while only keeping a small set of data blocks.

1. Introduction

Classifying network traffic according to its applications is important to a broad range of network areas including network monitoring, network management and traffic optimization, network security, traffic accounting, etc. Different from the traditional applications (http, email, ftp), new applications, especially P2P applications, usually use dynamic port numbers. The traffic classification technique based on native port has become less effective. However, the payload signature-based traffic classification technique [1,2] can achieve high accuracy. But the technique also has its limitations. It is ineffective in classifying encrypted traffic. Besides, a lot of P2P applications use proprietary protocols. Lacking open protocol specifications makes analyzing signatures and maintaining up-to-date signatures difficult.

Recently, machine learning algorithms which classify network traffic using flow statistical information [5–9] have been proposed. There are several challenges in classifying network traffic by using flow properties. First, the statistical characteristics used in classification are unstable since the delays and/or the packet loss ratios of the networks are dynamic. Second, flows belonging to different applications can have similar per-flow statistical characteristics. It is hard to distinguish these similar flows by using flow properties.

This paper proposes a novel approach to identify P2P traffic based on its data transfer behavior. The idea is based on the observation that a P2P peer uploads data to other peers after downloading it. The observation is first employed by Lu et al. [10]. In their method, the first k bytes of each packet in downloading flows are stored for each host. When the same content is found in uploading flows of the host, the flows associated with the content are classified as P2P flows. The partitioning scheme in their method is named the *head packet partitioning scheme* (i.e., searching shared data in the first few bytes of packets) in this paper. As shown in our experiments, the performance of the head packet partitioning scheme is poor in identifying some P2P applications.

The *content-based partitioning scheme* (i.e., searching shared data in the whole payload) is proposed in the paper to solve the problem. The scheme divides payloads of flows into data blocks (a data block is a contiguous content block of payload). The shared files and videos in P2P applications are usually divided into small data pieces during exchanges. For flows sharing the same data piece, the scheme can synchronize the boundary of the data piece, and extract the same part of the data piece as a data block. The scheme is generic and can be applied to most P2P applications.

Our contributions are as follows: First, experiments prove that the content-based partitioning scheme we proposed performs much better than the head packet partitioning scheme. Besides, *head tail partitioning scheme* (i.e., searching shared data in both the first and the last few bytes of packets), a simple enhancement of the head packet partitioning scheme, also can improve performance greatly, though not as good as the content-based partitioning scheme. Second, we have studied the performance impact of

* Corresponding author. Tel.: +86 6 260 3064; fax: +86 6 277 0753.
E-mail addresses: xuke@tsinghua.edu.cn (K. Xu), tigerCN7@gmail.com (M. Zhang), yemingjiang@csnet1.cs.tsinghua.edu.cn (M.J. Ye), dmchiu@ie.cuhk.edu.hk (D.M. Chiu), jianping@cernet.edu.cn (J.P. Wu).

0140-3664/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2010.01.005

some important issues. They are the size of the data block, the number of the data blocks to be stored, the data block replacement algorithms and the ratio of unobservable communications.

From the studies, we have drawn several conclusions. First, 256 bytes is a suitable size for data blocks. Second, the random replacement algorithm is suggested in replacing old data blocks. Third, with the random replacement algorithm, the method performs quite well while only keeping a small data block set (3 min). Last, even though the communications of a large fraction of peers (about 30%) are not observed, the performance degradation is rather small.

2. Related work

Payload signatures are useful in classifying network traffic [1,2]. Application signatures are the common strings in P2P protocols to identify P2P traffic, whereas our method focuses on the data being shared in P2P applications.

In addition to signature-based methods, other payload-based methods are also proposed. ACAS [3] uses the first *N* bytes of payload as input to train a machine learning model to classify flows. Levchenko et al. build several probabilistic models on payload, including the statistical model treating each *n*-byte flow distribution as a product of *N* independent byte distributions and the Markov process model which relies on introducing independence between bytes [4]. The models still employ frequently appearing bytes in application protocols to classify traffic since the random bytes in application data are meaningless in statistics.

The machine learning algorithms classify network traffic by using traffic characteristics [5–9], such as average length of packets and arrival interval of packets. The algorithms can be further summarized into supervised and unsupervised methods. Zander et al. compare several supervised algorithms, including Naive Bayes, C4.5 decision tree, Bayesian Network and Naive Bayes Tree [9]. They find that the classification precision of the algorithms is similar, but computational performance can be significantly different. McGregor et al. use the unsupervised expectation maximum algorithm to cluster the flows [6]. The experiment finds that the average precision of classification is very high, but some applications are difficult to distinguish.

The special communication patterns of applications are used in traffic classification. Karagiannis et al. study the communication pattern of P2P traffic in transport layer to identify P2P traffic [11]. BLINC attempts to capture the communication pattern of a host at three levels: the social level, the functional level and the application level [12]. Graphlet is used to describe the communication pattern of each application and classify network traffic.

It is hard to find a general communication pattern for all P2P applications. Hu et al. propose a method to build behavioral profiles of the target application which then describes the dominant communication patterns of the application [13]. The profiles of each application are built with data mining algorithms in the training phase. The data transfer behavior used in our method is general for all P2P applications, so our method does not need any training phases.

A similar work to ours has been presented by Lu et al. [10]. Their work opens up a new approach to identifying P2P traffic by inspecting data transfer behavior. But they only use the first k bytes of payload to find the shared data and the number of identifiable P2P applications is limited. We employ their ideas but propose a new payload partitioning scheme. Experiments show that our method is better and general for different kinds of P2P applications. Besides, we have studied the parameters affecting identification accuracy and examined the computation cost and memory overhead.

3. P2P traffic discovery with the content-based partitioning scheme

We first describe the content-based partitioning scheme used in our algorithm. Then we illustrate that this scheme is very effective because identification data blocks can be generated even when a small amount of data is shared between flows. At last, we propose a new traffic identification algorithm based on the content-based partitioning scheme.

3.1. Payload partitioning schemes

The basic idea is simple. The method inspects the P2P data transfer behavior on the host level. From the view of a host, flows can be classified as downloading flows and uploading flows. If a host downloads a data piece in a downloading flow, and then uploads it to other hosts in some uploading flows, these flows are classified as P2P flows.

The biggest obstacle is to find the same data pieces in different flows. One way is comparing the payloads of flows directly. But it is time-consuming. Besides, complete payloads of flows have to be saved in memory before comparing. It is quite difficult — and therefore, impractical — as an online process.

In our method, payloads of flows are divided into data blocks and then the signatures of the data blocks are compared. A payload partitioning scheme decides how to divide the payload into data blocks. The ideal result is that each data block is an exact data piece in a P2P application. For example, for two distinct flows in Fig. 1, the ideal case is that the generated data blocks in the first flow equal to data pieces 1, 5 and 7, and the generated data blocks in the second flow equal to data pieces 9, 4 and 1. Thus, the shared data (data pieces 1) of the two flows can be found.

However, a prerequisite to generating such ideal results is the detailed protocol information. As shown in Fig. 1, there are several challenges in locating the boundaries of data pieces. First, a flow can contain protocol fields with variable sizes. Second, the sizes of data pieces are variable in some P2P live streaming applications, where a data piece contains one second of video content. The size of the data piece is variable in most video coding schemes.

Three partitioning schemes are considered. The first one is the head packet partitioning used in [10]. If the packet size of a packet exceeds the threshold *S*, the first *S* bytes of the packet is extracted as a data block.

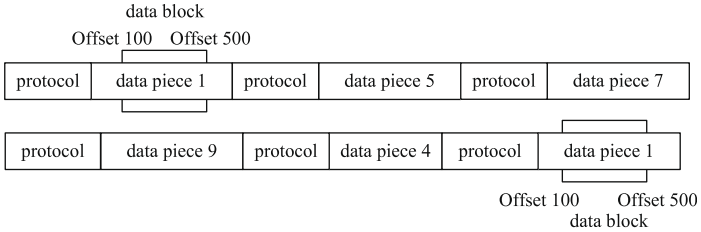


Fig. 1. Shared data piece detection.

```
Content-Based Partitioning Algorithm
void ContentBasedChunking (bytes, n, W, D, r)
1)   lastBoundary = 0
2)   for (int offset=0; offset+W < n; offset++)
3)     F = RabinFingerprint (bytes[offset,offset+W-1])
4)     if (F mod D == r)
5)       curBoundary = offset + W-1;
6)       dataPiece = bytes[lastBoundary, curBoundary]
7)       output dataPiece
8)       lastBoundary = curBoundary+1
9)     end if
10)  end for
```

Fig. 2. Content-based partitioning algorithm.

The second one is the head tail packet partitioning scheme, an enhancement of the previous one. If the payload size of a packet exceeds the threshold value S, the first S bytes of payload and the last S bytes of payload are extracted as two data blocks. If the target packets contain some protocol fields at the beginning or at the end, the scheme can skip them.

The last one is the content-based partitioning scheme. It has been used in saving bandwidth in network file systems [14] and automatically generating signatures of worms in security fields [15]. The scheme divides a flow into variable-size, non-overlapping data blocks. It works as follows. First, a pair of pre-determined integers D and r ($r < D$) are set. Then a W-width sliding window moves across the byte sequence. The window begins from the first W bytes in the sequence, and slides one byte at a time toward the end. At every position of the byte sequence, a fingerprint F is calculated according to the content in the current window. If F mod D equals r, the end of the window is a data block boundary (Fig. 2).

For example, in Fig. 3, the window size (W) is 5, D is 8 and r is 7. A 5-byte-width window moves across the byte sequence and fingerprints are calculated in each position. When the window reaches {abcde}, two positions can satisfy the condition. Two data blocks {tzynuns} and {ufabcde} are extracted.

The fingerprint is calculated with Rabin fingerprinting algorithm [16] which has a low collision rate. Using the pre-computed table, it is efficient to calculate the Rabin fingerprint [17].

The principle behind the content-based partitioning scheme is that, because the scheme determines the boundaries of data blocks based on the local content of payload in a small window, the boundaries can synchronize in the same data pieces. For example, in Fig. 4, there are two flows containing the same data piece (the black part) in different positions. In the first flow, there are two positions satisfying the condition. They locate at offset 100 and 500 from the beginning of the data sequence. In the second flow, the positions at the same offsets from the beginning of the data piece should also satisfy the condition, as the contents in the windows are the same. The same data block is extracted in both flows.

The content-based partitioning scheme can create data blocks with various sizes. Although the average size of blocks is D, the data blocks can be as short as several bytes. Short data blocks introduce many false positives, so the results are further filtered to only keep data blocks whose sizes exceed threshold S.

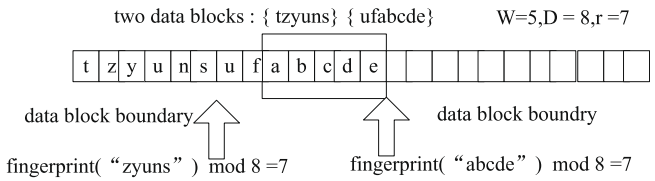


Fig. 3. Content-based partitioning scheme.

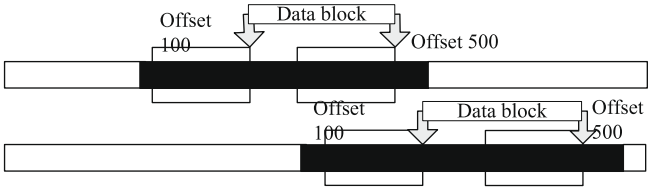


Fig. 4. Principle of content-based partitioning.

The value of D decides the average size of data blocks. If D is much smaller than S, many data blocks which are smaller than S will be generated and filtered. So D is equal to S in the algorithm. Since the performance is not sensitive to the value of r, we set r to $D-1$ in the algorithm.

The boundaries of data blocks are determined by the local content in the window. The window size should be much smaller than the size of data pieces to generate data blocks inside a data piece. However, a small window is sensitive to random content in flows. In the experiments, the window size is 32 bytes.

3.2. Effectiveness of the content-based partitioning scheme

In this section we show the effectiveness of the content-based partitioning scheme. Specifically, the relationship between the probability of generating a data block and the shared data length is studied.

Lemma 3.1. *If the length of shared continuous payload between two flows is L, and the content-based partitioning scheme generates a data block in one flow whose offset is larger than W (sliding window width, mentioned above), the same data block can be generated in the other flow with the partitioning method.*

Proof. Suppose the data sequences are S_1 and S_2 in the two flows. The starting indexes of the shared payload are x_1 and x_2 separately. Then we have

$$S_1(x_1+i) = S_2(x_2+i), \quad \forall i \in \{0,1,\ldots,L-1\}. \tag{1}$$

We can get a data block with the content-based partitioning scheme and the Rabin fingerprinting algorithm. Suppose the generated data block in flow S_1 is $S_1[x_1^b, x_1^b+L_b-1]$, whose length is L_b, then $x_1+W \leqslant x_1^b,\ x_1^b+L_b-1 \leqslant x_1+L-1$. Furthermore,

$$rabin(S_1[x_1^b-W, x_1^b-1]) \mod D = r \tag{2}$$

and

$$rabin(S_1[x_1^b+L_b-W, x_1^b+L_b-1]) \mod D = r. \tag{3}$$

By Eqs. (1)–(3), we have

$$S_2[x_1^b-W+x_2-x_1, x_1^b-1+x_2-x_1] = S_1[x_1^b-W, x_1^b-1], \tag{4}$$

$$S_2[x_1^b+L_b-W+x_2-x_1, x_1^b+L_b-1+x_2-x_1] = S_1[x_1^b+L_b-W, x_1^b+L_b-1]. \tag{5}$$

And finally,

$$rabin(S_2[x_1^b-W+x_2-x_1, x_1^b-1+x_2-x_1]) \mod D = rabin(S_2[x_1^b+L_b-W+x_2-x_1, x_1^b+L_b-1+x_2-x_1]) \mod D = r. \tag{6}$$

That is to say, S_2 also generates two data block boundaries and the payload content between them is the same as that in S_1, which completes the proof. □

Lemma 3.1 shows that the same data blocks can be generated between different flows regardless of the different offsets of the

shared data. So if the same data block is found in different flows, we can demonstrate that they have shared data.

Lemma 3.2. *The probability of generating a data block, whose offset is no less than W, for the payload of length L with the Rabin fingerprinting algorithm is*

$$P_r = \begin{cases} 0 & \text{if } L \leqslant W, \\ 1 - \frac{L+D-W}{D}\left(\frac{D-1}{D}\right)^{L-W} & \text{if } L > W. \end{cases}$$

Proof. The content-based partitioning scheme generates a Rabin fingerprint in each byte boundary. A data block cannot be generated if there are less than two data boundaries. Obviously no signature of length larger than W can be generated if $L \leqslant W$. If $L > W$, the fingerprint F of a data block boundary must satisfy $(F \bmod D = r)$. Suppose the payload content is randomly generated (As everything can be shared between peers, the hypothesis is reasonable.), then the probability that a piece of payload is a data block boundary is $(p = \frac{1}{D})$. Furthermore, as the Rabin fingerprints are calculated by hashing, the probability of a data block boundary of different positions can be assumed independent. So the probability $P_r(i)$ to generate i data block boundaries follows the binomial distribution with parameters $(L - W + 1, 1/D)$, which is

$$P_r(X = i) = C^i_{L-W+1}\left(\frac{1}{D}\right)^i\left(\frac{D-1}{D}\right)^{L-W+1-i}. \tag{7}$$

And the probability to generate at least two data block boundaries with payload length $L - W + 1$ is

$$P_r(i \geqslant 2) = 1 - P_r(0) - P_r(1).$$

The probability to get a data block is

$$P_r(i \geqslant 2) = 1 - \frac{L+D-W}{D}\left(\frac{D-1}{D}\right)^{L-W} \quad \square \tag{8}$$

Lemma 3.2 gives the probability of generating a data block with the given data length.

Theorem 3.3 (Ye's Theorem). *Let the length of payload shared between different flows be L, the probability to generate at least one identifying data blocks with the content-based partitioning scheme is*

$$P_r = \begin{cases} 0 & \text{if } L \leqslant W, \\ 1 - \frac{L+D-W}{D}\left(\frac{D-1}{D}\right)^{L-W} & \text{if } L > W. \end{cases}$$

Proof. Lemma 3.2 says that the probability of generating at least one data block is shown in Eq. (8). And Lemma 3.1 shows that every data block from one flow will definitely appear in another flow, so the probability of generating data blocks between flows is the same as that in Lemma 3.2. $\square$

Like Lemma 3.2, Theorem 3.3 gives the probability of generating data blocks if payload of the given length is shared. The theorem says that no data block can be generated if the shared data length is small, which filters a lot of noise. But if the shared data length is tentatively large, it is probable to generate data blocks. So the content-based partitioning scheme is both stable and effective.

3.3. Algorithm

The details of the algorithm are as follows: For each host, the algorithm keeps a hash set, which is called the downloading set, to save data blocks. Payloads of flows are divided into data blocks. Data blocks of each downloading flow and flow identifiers (a flow identifier is the 5-tuple: source IP address, source port, destination IP address, destination port, and protocol) are saved in the corresponding downloading set. To save the memory capacity, signatures of data blocks (Rabin fingerprint) are calculated and saved instead of the original data blocks. If the size of the downloading set exceeds the limitation, data block replacement is applied. Data blocks of each uploading flow are checked whether they have been saved in the corresponding downloading set already. If a data block of the uploading flow has been saved in the downloading set, the uploading flow, the downloading flow and their reverse flows are identified as P2P flows.

A flow has two roles. One is the downloading flow of the target host. The other is the uploading flow of the source host. The data blocks of the flow are saved in the downloading set of the target host and checked in the downloading set of the source host.

The online process of the algorithm is illustrated in Fig. 5. When a packet arrives, it is first partitioned into several data blocks. And then, the signatures of the data blocks are calculated. Finally the signatures are saved into the downloading set of the destination host and checked in the downloading set of the source host.

A more detailed algorithm description is shown in Fig. 6. Every time a packet arrives, flow reassembly is done by concatenating the payload of the packet to the remaining payload stored. The content-based partitioning scheme is applied to each byte offset to calculate data block boundaries. If a data block boundary is found, a valid data block is generated. Data blocks too small are dropped. The Rabin fingerprint of the remaining data block is calculated. And if it is found in the downloading set of the source host, the flow is identified as P2P flow. The fingerprint is added to the downloading set of the destination host if not found. Necessary replacement is done if the downloading set is full. In the perspective of algorithm performance, Rabin fingerprinting is the most time-consuming since a fingerprint is calculated for each offset. In addition, about every S bytes, a data block is found and its Rabin fingerprint is calculated. Thus, if the total packet length is Tot bytes, the number of Rabin fingerprinting operations is about $Tot + Tot/S$.

4. Evaluation

We study the parameters affecting identification accuracy in this section. Computation cost and memory overhead are also examined.

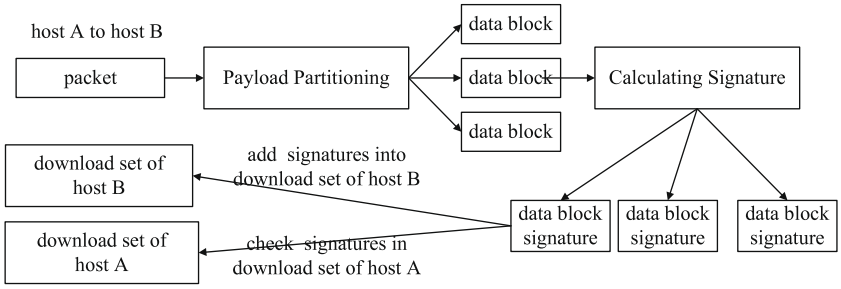


Fig. 5. Online process of the algorithm.

```
P2P Data Transfer Behavior Inspecting Algorithm
void P2PTrafficIdentify (flow, packet, src, dst)
1)   payload = concatenate (payload, packet) //flow reassembly
2)   n = payload length
3)   lastBoundary = 0
4)   for (int offset=0; offset+W < n; offset++)
5)     F = RabinFingerprint (bytes[offset,offset+W-1])
6)     if (F mod D == r)
7)       curBoundary = offset + W-1;
8)       dataPiece = bytes[lastBoundary, curBoundary]
9)       lastBoundary = curBoundary+1
10)        if dataPiece size < S //data block threshold
11)          continue
12)        F2 = RabinFingerprint (dataPiece)
13)        if F2 in the block set of host src
14)          output P2P flow
15)          return
16)        else
17)          insert F2 to the block set of host dst
18)          remove a data block if it's full
19)        end if
20)    end if
21)  end for
22)  payload = payload (lastBoundary, end)
```

Fig. 6. Identifying P2P traffic by inspecting the data transfer behavior.

4.1. Experiments setup

Two metrics are used [20]. The first one is the true positive (TP) rate. It is used to measure the traffic fraction that can be recognized by the algorithm out of the traffic belonging to the given application.

$$\mathrm{TP} = \frac{\text{traffic classified as the application}}{\text{Total application traffic}}. \quad (9)$$

The second one is the false positive (FP) rate. It is used to measure the traffic fraction which is not really produced by a given application out of the traffic classified as the application.

$$\mathrm{FP} = \frac{\text{Non-application traffic classified as the applciation}}{\text{Total traffic classified as the application}}. \quad (10)$$

They are important metrics especially when flow type identification is used in traffic management. For example, if network operators differentiate the service qualities according to the flow types, high false positive or low true positive rates can make high priority traffic suffer from performance degradation. A good algorithm should have low false positive and high true positive rates.

As shown in Table 1, a lot of popular P2P applications are evaluated in the experiments. They are classified into three types: P2P file sharing, P2P live streaming and P2P Video on Demand (VoD).

To evaluate the method, two kinds of traffic traces are captured and replayed: pure application traces and mixed application traces. The former traces are captured from a host which only has the given application running on it. Each P2P file sharing application trace contains the traffic generated by downloading several files. Each P2P live streaming or P2P VoD application trace contains the traffic generated by watching a video. The time slot of each trace file varies from half an hour to one hour. The traces are only used to evaluate the true positives.

The mixed traces are captured from a host running various applications. Each trace contains the traffic generated by a P2P application and some non-P2P applications, such as HTTP, FTP, SMTP and POP3. P2P applications in them are labeled by their payload signatures. In each mixed trace, the P2P traffic accounts for 30–40% of the total traffic. The traces are used to evaluate both the true positives and the false positives. We have generated mixed traces with two P2P applications. They are BitTorrent and PPLive.

Table 1
P2P applications.

Type	Applications
P2P file sharing	BitTorrent, Emule
P2P live streaming	PPLive, PPStream, TVAnt, FeiDian, PPMate, SinaLive, TVULive
P2P video on demand	PPLive VoD, XL VoD, PPStream VoD

4.2. Comparing partitioning scheme

First, the true positive of different partitioning schemes is studied by applying them on the pure application traces. In the experiments, the threshold S is 256 bytes. To study the best true positive that each scheme can achieve, the size of the downloading set is unlimited.

True positives are shown in Fig. 7. Among the three partitioning schemes, performance of the head packet partitioning scheme is the worst. Quite a few P2P applications can't be recognized by the scheme. Performance of the head tail packet partitioning scheme is almost as effective as the content-based partitioning scheme, except for the FeiDian live streaming application. If the applications have protocol fields in both the head and the tail of the packets, the packet-based partitioning schemes do not work. The content-based partitioning scheme is more generic.

For most applications, the true positives in bytes are more than 90%, but it is only 40% for the Emule application. There are two reasons. First, the downloaded files which are then uploaded by Emule are older than those by BitTorrent. Second, Emule transfers a part of a file with 1300 bytes in a separate packet each time to avoid fragmentation. The start position of the part in the file is specified in a request message. The data pieces exchanged in BitTorrent protocol are globally divided, but the data pieces exchanged in Emule protocol are not. It is more difficult to find the shared data pieces in flows of the Emule application.

4.3. Parameter tuning

In this section, we study the parameters affecting identification accuracy.

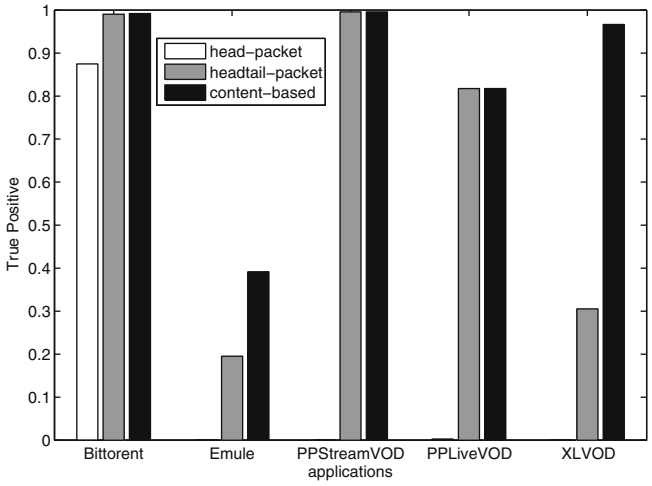
4.3.1. Data block size

The threshold S in the content-based partitioning scheme is important. The effect of threshold S is evaluated in Fig. 8. Parameters W and D are set to 32 and 256, respectively. The size of the downloading set is unlimited.

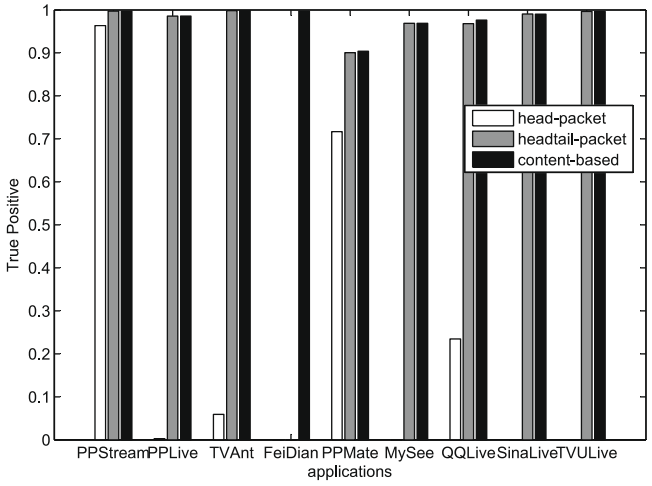
The consequences of a smaller threshold S are: the generation of more data blocks; higher memory consumption and increased false positives. However, the probability of finding the shared data pieces decreases as the threshold S increases.

True positives of most applications do not decrease significantly when S is smaller than 1024 bytes, except the XLVoD application. It implies that the data block size of most P2P applications are smaller than 1024 bytes.

To further study the effect of parameters W and D on data block size, we make another study with mixed traffic traces. We run through the traces with different W and D values. Fig. 9 analyzes the size of the generated data blocks and shows that smaller data blocks ($<D$) can be more probably generated while large blocks

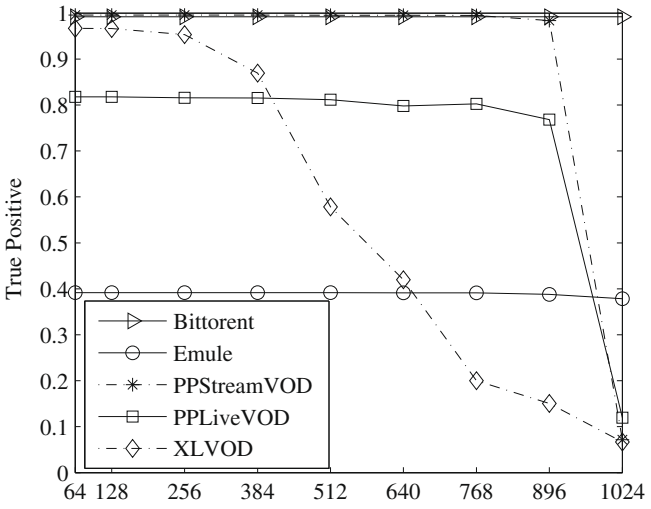


(a) Group I: File Sharing, VoD

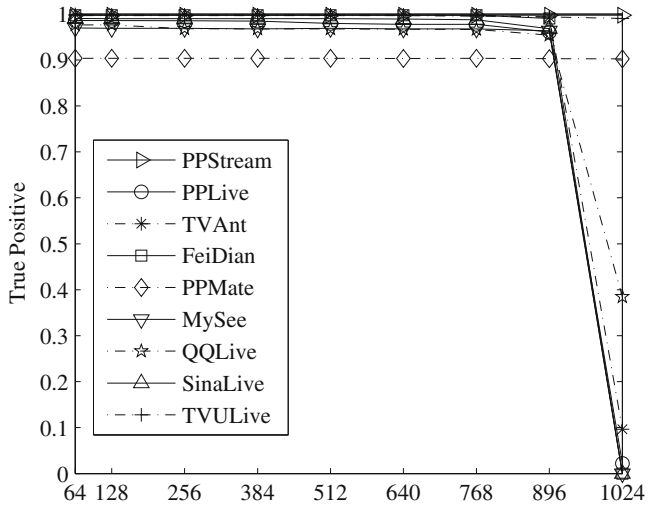


(b) Group II: Live Streaming

Fig. 7. True positives for different applications.



(a) Group I: File Sharing, VoD



(b) Group II: Live Streaming

Fig. 8. True positives with different threshold *S*.

(>4 * *D*) are rare. Parameter *D* has an obvious effect on data block size. When in real environments, *D* can be adjusted tentatively large to increase average data block size and decrease processing overhead. On the other hand, window size *W* does not show great impact on data block size (Fig. 9(a) and (b)). But it should neither be set too large (>*D*) nor too small (a few bytes) in real cases.

Fig. 9 also explains why the true positive decreases when *S* is large. If *S* is too large, chances to generate a data block decrease and even become impossible. The threshold *S* of 256 bytes (for *D* = 256) is suggested in the algorithm and used in the following experiments.

4.3.2. Downloading set size

The algorithm keeps recent data blocks in a downloading set for each host. Saving data blocks for an extended period can improve the true positive, but it requires a lot of memory.

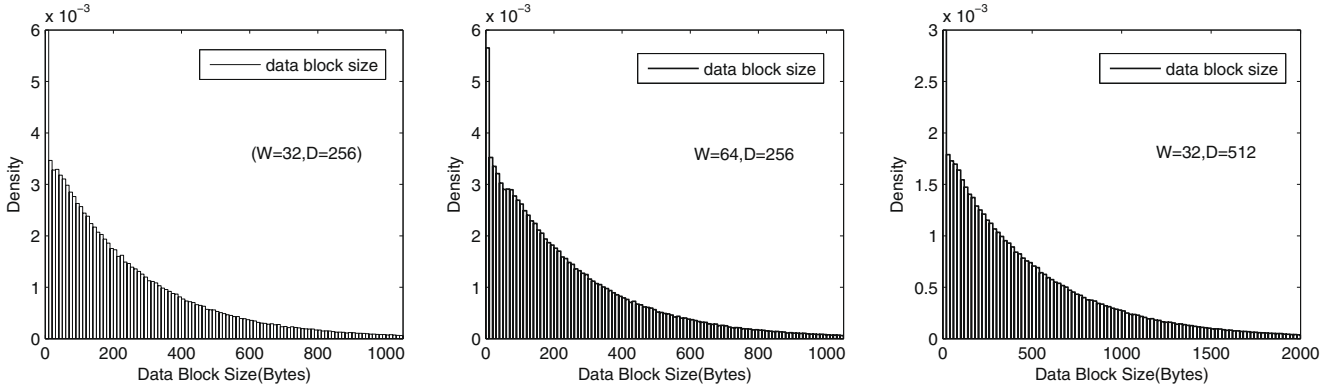
The effect of the downloading set size is shown in Fig. 10. The threshold *S* is 256. The size of the downloading set is measured in minutes. The size of 1 min means that the downloading set will keep data blocks generated in the last 1 min.

Because peers exchange the video content in a small time window in P2P live streaming applications, the true positives of P2P live streaming applications are much better than P2P file sharing and P2P VoD applications for small downloading sets. For P2P file sharing and P2P VoD applications, downloaded data pieces can be uploaded after a long time. Keeping the most recent data blocks requires a large time window for the applications.

The performance can be further improved using other replacement policies instead of the current first-in, first out (FIFO) policy. There are two other data block replacement policies to consider. They are least recently used (LRU) and random replacement (RANDOM).

We have evaluated the performance of several P2P applications (Fig. 13). The *x*-axis is the size of the downloading set and the *y*-axis is the normalized true positive. It is calculated by dividing the current true positive by the true positive of the unlimited downloading set. If the value is 1, it implies that the performance of a small downloading set is as good as the one using the unlimited downloading set.

The experimental results in Fig. 10 indicate that a one-minute time window is large enough for P2P live streaming applications. We also tried other replacement policies and each of the three policies works almost the same for them. But for P2P file sharing and P2P VoD applications, the random replacement algorithm works much better than the other algorithms (Fig. 13). A downloading set of a three-minute time window is large enough for all P2P applications using the random replacement policy.

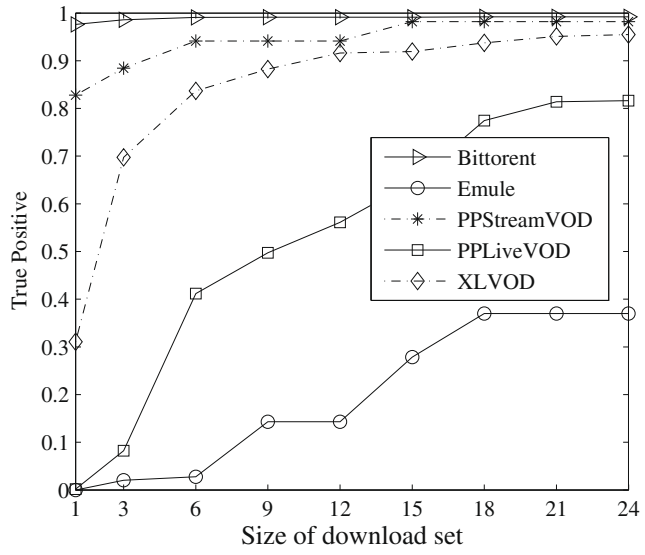


(a) Distribution of Data Block Size (W=32,D=256)

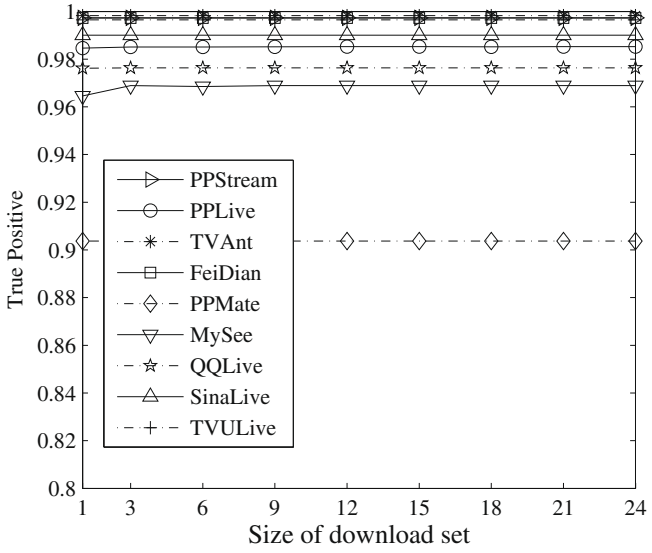
(b) Distribution of Data Block Size (W=64,D=256)

(c) Distribution of Data Block Size (W=32,D=512)

Fig. 9. Effect of parameters *W* and *D* on data block size.



(a) Group I: File Sharing, VoD



(b) Group II: Live Streaming

Fig. 10. True positives with different downloading set sizes.

The idea of the random replacement policy is that, old data blocks are more likely to be kept by the algorithm than other algorithms. Keeping the old data blocks can improve true positives for P2P file sharing and P2P VoD applications. Besides, the algorithm has a positive bias to big flows, which can also improve true positives in bytes. A flow transferring more traffic has a higher probability to be recorded and identified in random replacement.

4.4. Algorithm performance

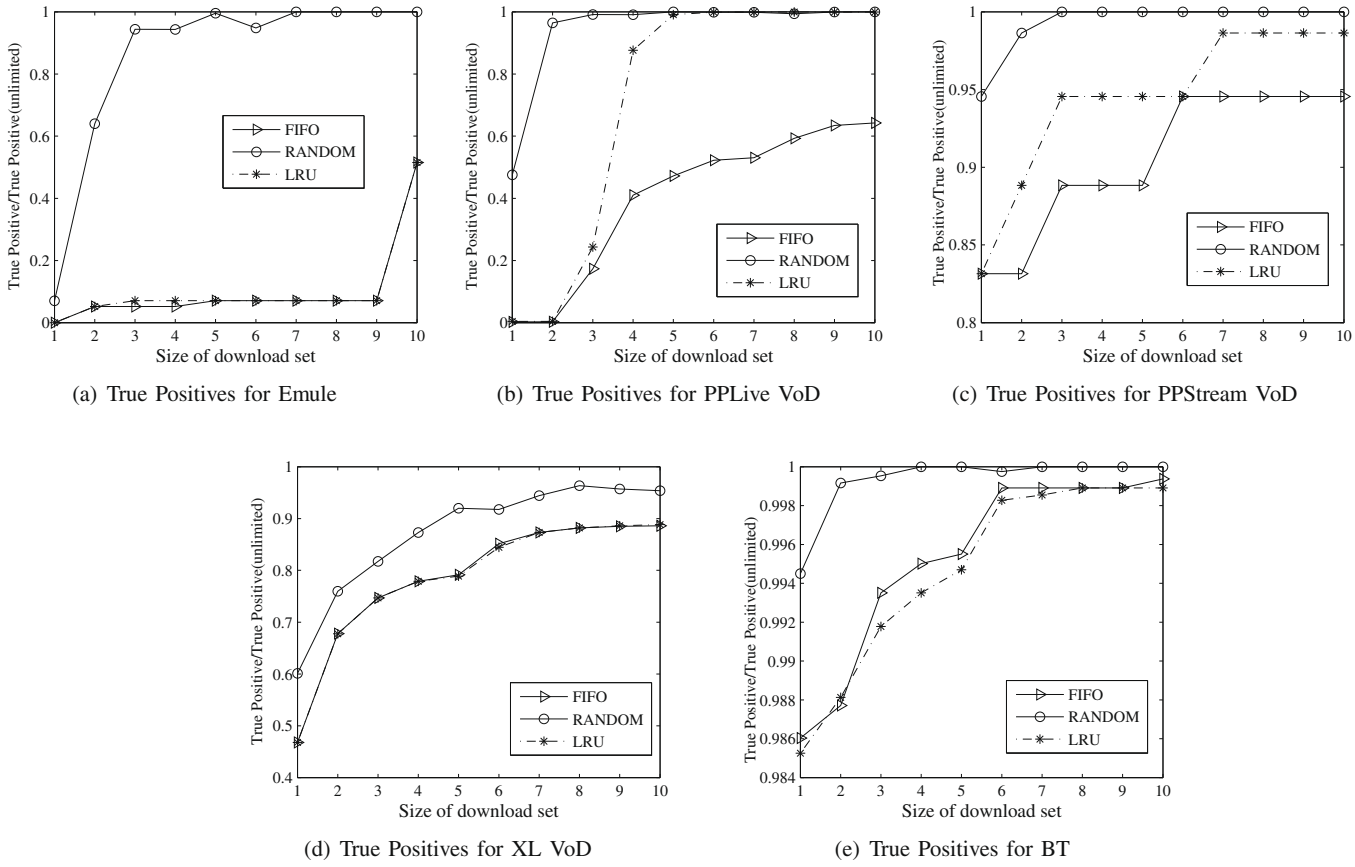
The computation and memory overhead are important in identifying P2P traffic online with our method. In this section, we first estimate the throughput by counting the CPU cycles. Then we discuss memory usage.

4.4.1. Throughput

We break the algorithm into small operations and count the CPU cycles (on a notebook with 1.60 GHz processor speed, 512 MB 798 MHz memory) for each operation. An approximately 600 MB mixed traffic trace is used. Because of variable packet sizes and data block sizes, the average processing time and its deviation is reported. The results are shown in Table 2. Specifically, the most expensive operations are retrieved and evaluated. There are mainly three components: payload concatenation (i.e., flow reassembly), Rabin fingerprinting and hash set operation. The overall processing time for the algorithm is also shown.

Table 2
Processing time (in microseconds).

	Steps in Fig. 6	Mean	Standard deviation
Payload concatenation	1), 22)		
Per packet		0.89	0.117
Per byte		0.0024	–
Rabin fingerprinting			
First fingerprint (32 bytes)	5)	0.336	0.073
Increment (per byte)	5)	0.031	–
Data block fingerprint	12)	2.026	0.177
Hash set operation			
Hash set query	13)	0.78	0.110
Insert in hash set	17)	2.465	0.197
Remove in hash set	18)	2.2	0.186
Overall average processing cost (per byte)			
Case 1: fingerprinting by software		0.498	1.827
Case 2: fingerprinting by hardware		0.043	0.523



(a) True Positives for Emule (b) True Positives for PPLive VoD (c) True Positives for PPStream VoD

(d) True Positives for XL VoD (e) True Positives for BT

Fig. 11. True positives of different unobservable host ratios.

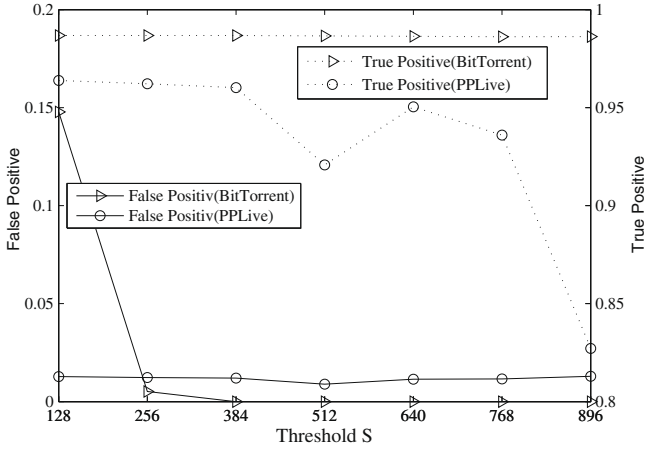


Fig. 12. False positives and true positives for BitTorrent and PPLive.

Rabin fingerprinting is the most expensive operation and calculating it for data block boundaries dominates the overall cost. Fingerprinting data blocks is also very expensive, so as the data block set maintenance operations. But these operations are unusual since small data blocks are filtered.

Table 2 shows that on average it takes 0.498 μs for each byte. This equals to 0.06 μs per bit, or a 20 Mbps line rate. It can be improved by accelerating the Rabin fingerprinting process with hardware. Once the irreducible polynomial is determined, a precomputed table can be generated and fingerprinting is actually looking up tables and doing logical computation with input characters [17]. A dedicated SRAM can meet our requirements and the performance is expected to increase around 20–50 times. The overall average processing cost can be lowered to 0.043 μs, which is about 200 Mbps line rate. It is the lower bound, and performance can be much better in real environments. First, parallelism can be explored and faster hardware can be employed. Second, not all the data blocks are fingerprinted for P2P traffic. Because only data blocks for unidentified flows are processed in real environments, and P2P traffic usually dominates the overall traffic volume, there are far less data block fingerprinting operations. In addition, data block set maintenance can be efficient since keeping data blocks for a few minutes is enough. If 60% of the overall traffic is P2P, throughput can reach about 500 Mbps. So the throughput is expected to reach more than 500 Mbps in normal traffic environment if faster hardware is used.

4.4.2. Memory consumption

Memory consumption is also affordable in our algorithm. Memory is used mainly for two purposes. First, in the payload partitioning, the byte level partitioning schemes have to keep some intermediate information for each flow. For example, to update the fingerprint in the content-based partitioning scheme, the fingerprint and the content of the last window are saved for each flow. The intermediate information is quite small and 40 bytes are enough to keep the intermediate status. Suppose there are 1 M concurrent flows, only 40 MB memory is needed.

Second, the signatures and flow identifiers of the data blocks are saved. Suppose the size of a signature is 16 bytes and the size of the flow identifier is 4 bytes (an index in the flow table is used instead of original 5-tuple), 20 bytes are needed to record a data block. If a data block is as large as 512 bytes, a bidirectional 1 Gbps link with

50% link utilization can generate at most 1 Gb/8/512 $*$ 20 = 5 MB records per second. It costs about 300 MB of memory for saving 1-min records and 1.46 GB of memory for saving 5-min records.

The experiments show that keeping data blocks for several minutes is enough. So the memory consumption is affordable in current hardware capabilities. Besides, because a lot of small data blocks are not saved, the simple estimation causes the results to be the upper bounds. Other methods can further reduce memory consumption. For example, flows which are too small or too short are unnecessary to be saved since they are unlikely to be P2P downloading flows.

4.5. False positive

The mixed traces are used to evaluate false positives for BitTorrent and PPLive. The size of the downloading set is unlimited in the experiment. The results are shown in Fig. 12. The *x*-axis is the threshold *S* while the left *y*-axis is the false positive and the right *y*-axis is the true positive.

As the result, a threshold of 256 bytes can help to guarantee a low false positive while non-P2P applications are transferring random data. But some behaviors of non-P2P applications may lead to false positives. For example, people forward email they received. Some methods can be used to eliminate these false positives [11]. Since a P2P host always has a service port, we can further identify the P2P {IP, port} pairs which associate with many identified flows. The flows which are not associated with a P2P {IP, port} pair are filtered to eliminate false positives. For other tricky data transfer behaviors, such as viewing comments posted earlier, checking in changes and checking out immediately when maintaining software, similar methods can be employed to achieve best identification accuracy.

4.6. Deploying place

In previous experiments, all the communications of the host can be observed and analyzed. When deploying the algorithm in the gateway of an institute or an edge router, the communications inside the intra-network are unobservable. The performance of absent communications is studied in the experiment. The threshold *S* is 256 bytes, the size of the downloading set is 3 min and the random replacement policy is used in the experiment.

The results are shown in Fig. 11. The *x*-axis is the fraction of unobservable hosts. For example, 0.2 means that flows between the monitored host and 20% of the other hosts are filtered. The missed hosts are selected randomly and the flows are removed from the original trace. The *y*-axis is the true positive on the filtered trace.

The results indicate that even though a large fraction of hosts (about 30%) are unobservable, the algorithm can still achieve a high true positive. It also implies that the algorithm can work well even when the deploying place is not close to the hosts being inspected, such as the gateways of large institutes and edge routers.

4.7. Discussion

Our payload-based method focuses on the same data being shared in P2P applications, so it is ineffective in classifying encrypted traffic. Besides, the P2P applications using network coding [18] are also undetectable with our method. We argue that encryption and network coding pose a burden on P2P application developers, so the mainstreaming P2P applications are still transferring data without any transformation. Non-payload-based methods are more efficient to identify encrypted traffic. There are some studies using machine learning algorithms to identify encrypted traffic [19].

Besides, our method assumes that peers will upload the received data to others. But some peers may only download data without providing data to others. This causes some P2P traffic cannot be identified. But this case is rare since incentive mechanism for providing data is becoming popular. In addition, most users use default settings for their P2P software, which usually enables sharing data to others by default.

5. Conclusion and future work

The paper proposed the content-based partitioning scheme to discover shared data and identify the P2P data transfer behavior. Compared with head packet partitioning scheme and head tail packet partitioning scheme, content-based partitioning scheme is generic and more accurate. Some important issues are also studied by experiments. The experiments show that our algorithm can achieve a high true positive and a low false positive while only keeping a rather small data block set with random replacement policy. In addition, our method can scale to more than 500 Mbps line rate if the fingerprinting process is implemented in hardware, which is satisfiable for practical deployment for edge routers and local area networks.

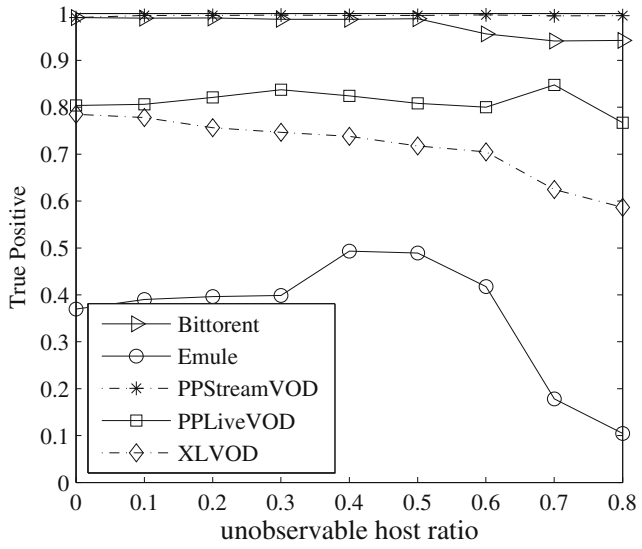


Fig. 13. True positives for different kinds of applications.

Our future work is to distinguish different P2P application flows by using their relationships in data exchange and flow properties. Our algorithm has introduced several new traffic characteristics including data block size and data block hit interval. We plan to improve P2P identification accuracy with these features as well as normally used flow characteristics. Furthermore, our method may fail if users don't upload the data they downloaded. We plan to extend our algorithm by inspecting unidirectional flows to overcome the affections. We are also interested in improving our algorithm with sampling to adapt to high speed backbone network environment.

Acknowledgement

This research is supported by NSFC-RGC Joint Research Project (20731160014), 973 Project of China (2009CB320501), 863 Project of China (2008AA01A326), NSFC Project (60970104, 60873253), Ericsson-Tsinghua Joint Research Project and Program for New Century Excellent Talents in University.

References

[1] A.W. Moore, K. Papagiannaki, Toward the Accurate Identification of Network Applications, Passive and Active Measurements Workshop, Boston, MA, USA, 2005.
[2] S. Sen, O. Spatscheck, D. Wang, Accurate, scalable in-network identification of P2P traffic using application signatures, in: Proceedings of ACM WWW'04, 2004.
[3] P. Haffner, S. Sen, O. Spatscheck, D. Wang, ACAS: automated construction of application signatures, in: Proceedings of the 2005 ACM SIGCOMM Workshop on Mining Network Data, ACM, New York, 2005, pp. 197–202.
[4] J. Ma, K. Levchenko, C. Kreibich, S. Savage, G.M. Voelker, Unexpected means of protocol inference, in: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, ACM, New York, 2006, pp. 313–326.
[5] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, C. Williamson, Semi-supervised network traffic classification, in: Proceedings of the ACM SIGMETRICS, 2007.
[6] A. McGregor, M. Hall, P. Lorier, J. Brunskill, Flow clustering using machine learning techniques, PAM (2004).
[7] S. Zander, T. Nguyen, G. Armitage, Self-learning IP traffic classification based on statistical flow characteristics, PAM (2005).
[8] A.W. Moore, D. Zuev, Internet traffic classification using bayesian analysis techniques, in: Proceedings of the ACM SIGMETRICS, 2005.
[9] N. Williams, S. Zander, G. Armitages, A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification, SIGCOMM Computer Communication Review 36 (5) (2006) 5–16.
[10] X. Lu, H. Duan, X. Li, Identification of P2P traffic based on the content redistribution characteristic, Communications and Information Technologies (2007).
[11] T. Karagiannis, A. Broido, M. Faloutsos, K. Claffy, Transport layer identification of P2P traffic, in: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement, ACM, New York, 2004, pp. 121–134.
[12] T. Karagiannis, K. Papagiannaki, M. Faloutsos, BLINC: multilevel traffic classification in the dark, ACM SIGCOMM, Philadelphia, PA, 2005.
[13] Y. Hu, D.M. Chiu, J.C.S. Lui, Application identification based on network behavioral profiles, IEEE IWQoS (2008).
[14] A. Muthitacharoen, B. Chen, D. Mazieres, A low-bandwidth network file system, in: Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01), Banff, Canada, pp. 174–187.
[15] H. Kim, B. Karp, Autograph: toward automated, distributed worm signature detection, in: Proceedings of the USENIX Security Symposium, Diego, 2004, pp. 271–286.
[16] M.O. Rabin, Fingerprinting by Random Polynomials, Tech. Rep. TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.
[17] A.Z. Broder, Some applications of Rabin's fingerprinting method, Sequences II: Methods in Communications, Security, and Computer Science, Springer-Verlag, New York, NY, 1993. pp. 143–152.
[18] C. Gkantsidis, J. Miller, P. Rodriguez, Comprehensive view of a live network coding P2P system, in: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, IMC'06, ACM, New York, NY, pp. 177–188.
[19] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, P. Tofanelli, Revealing Skype traffic: when randomness plays with you, ACM SIGCOMM Computer Communication Review 37 (4) (2007) 37–48.
[20] L. Salgarelli, F. Gringoli, T. Karagiannis, Comparing traffic classifiers, SIGCOMM Computer Communication Review 37 (3) (2007) 65–68.