A P2P Application Signatures Discovery Algorithm

Lijuan Duan[a,1], Lei Han[b], Yanfeng Yu[a] and Jian Li[a]

[a]College of Computer Science and Technology, Beijing University of Technology, Beijing, China.
[b]College of Computer Science, Polytechnic Engineer School of Tours University.

Abstract. In this paper, w**e** introduce a P2P application signatures discovery algorithm based on the combination of sequence mining with digital search tree. At the same time, we develop an application which contains two functions. One is the pre-treatment of packages, and the other is the discovery of application signatures. Finally, we use the real packages, which are snatched from the network, to gain effective application signatures**.**

Keywords. P2P application signatures discovery, sequence mining, digital search tree

1 Introduction

P2P is a kind of distributed network, where the participants share resources. P2P applications are now extending rapidly in the Internet. The current P2P traffic capacity is even more than traditional web applications, occupying above the half of the total network capacity. Various security problems will soon face with the huge developing range of P2P traffic capacity. We need to identify P2P system for real-time, in order to discovery and monitor P2P networks and block malicious information. There are two ways which commonly used to identify P2P system. The one method recognizes P2P application according to P2P network transmission behavior characteristic, which does not analyze the payloads [1]. The other method detects the payloads and finds signatures which match with known P2P application signatures. In order to hide its flow and avoid being detected based on ports, some P2P applications often transform the ports. As more and more P2P applications can operate on any port, the method based on the port number identification is no longer valid in the identification of P2P traffic [2]. Therefore, all data packet must be carried on the depth identification. It is said that the

[1] Associate Professor, College of Computer Science and Technology, Beijing University of Technology, 100 Pingleyuan, Chaoyang District, 100022,Beijing, China; Tel: +86 (10) 6739 6063; Fax: +86(10) 6739 1742; Email: ljduan@bjut.edu.cn

payloads of transport protocol TCP should be detected. So it can be judged whether the packets contain the sample signatures. That is the main idea of Deep Package Identification (DPI) based on application signatures.

2 Application Signatures Discovery Algorithm

The characteristic of P2P application signatures is high frequent and fixed position. Key Tree data structure is used to calculate the frequency and location of application signatures. Meanwhile, the sequence mining algorithm which can discovery high frequent sequence is adopted in order to get application signatures.
In this section, first we give a brief view of digital search tree and sequence mining, and then we introduce the combination algorithm.

2.1 Digital Search Tree

Key Tree is also called Digital Search Tree. It is a tree with more than 2 degrees. Each node in the tree contains an element which is a character of string. The characters in each path from the root to the leaf node represent a string. The special symbol '$' in leaf node indicates the end of the string. There is variable in the leaf node for counting the frequency.
Key Tree has characters as following.
(1) Each character of the string is distributed in the path from the root node to the leaf node. Therefore, the size of keyword sets is irrelevant to depth of the tree.
(2) Key Tree is an ordered tree. Each internal node x stores an element such that the elements stored in the left brother node of x are less than elements stored in the right brother node of x. It is assumed that symbol '$' is less than any other symbol.
There are two storage structures to represent Key Tree. The first one is child-brother list (double list). The second one is multilinked list (Trie tree). This paper selects the first one to implement Key Tree.

2.2 Sequence Mining

Sequence mining is concerned with finding statistically relevant patterns between data examples where the values are delivered in a sequence. A sequence is an order list of itemset. It is denoted as $\alpha = \alpha_1 \to \alpha_2 \cdots \to \alpha_n$, where α_i is a item. The length of α is n. A sequence with k items is called k-sequence. A sequence $\alpha = \alpha_1 \to \alpha_2 \cdots \to \alpha_n$ is contained in another sequence $\beta = \beta_1 \to \beta_2 \cdots \to \beta_m$, if there exist integers $i_1 \le i_2 \cdots \le i_n$ such that $\alpha_1 \subseteq \beta_{i_1}$, $\alpha_2 \subseteq \beta_{i_2}$,..., $\alpha_n \subseteq \beta_{i_n}$ [3]. For example the sequence ($B \to AC$) is a subsequence of ($AB \to F \to ACE$), since $B \subseteq AB$ and $AC \subseteq ACE$. A frequent sequence is maximal if it is not a subsequence of any other frequent sequence. The problem of mining sequential patterns is to find the high frequent sequences among all sequences that have a certain user-specified minimum support. Agrawal [3] presents AprioriAll to

mining sequence pattern. It is a three-phase algorithm: first finds all itemsets with minimum support, transforms the database, and then finds sequential pattern. But this algorithm does not handle time constraints. GSP [4] explores a candidate generation and test approach to reduce the number of candidates to be examined. The previously developed sequential pattern mining methods are used in customer database, DNA database and etc.

2.3 Combination of Sequence Mining with Digital Search Tree

For P2P application, we wish to find the sequence pattern in payloads. The difference with traditional sequence mining application is that the location information must be considered. Some characteristics which appears in specific positions are application signatures, while appears in other positions are general data. Based on the peculiar circumstance, this paper designs a new algorithm, which uses the idea of sequence mining and the statistical function of Key Tree to discovery application signatures of network packets. Since the algorithm is very similar to Apriori algorithm, it will face the same problem as Apriori algorithm. It needs a scan of the original database for each pass, so there is a lot of I/O operation. The efficiency of the algorithm will be affected. Key Tree will exactly solve the problem of massive I/O operates. It only needs one time I/O operation to read payload, and the subsequence selection can obtain data from Key Tree.
As a result of using Key Tree's structure, data is compressed. When large amounts of data stored in external storage is added to the tree, the memory space occupied is smaller than external storage. Meanwhile, we can also recover the original data without information lost. The statistical function of Key Tree can be used directly, so we need not to design an additional statistic program. The implementation process will discover frequent itemsets, which similar to steps of traditional sequence mining. First it finds the frequent 1-sequences L_1, which are in the same position within different payloads. Then it joins L_1 into C_2, the set of candidate 2-sequences. It deletes the all sequences $c \in C_2$ such that some 1-subsequences of c are not in L_1, and gets L_2. Using the same method, it tries until generate C_k, which turns out to be empty. Then it outputs all k-sequences.

2.3.1 Data preparation
Ethereal is used to capture packet from network and filter some unrelated packet, so that some packets generated by some kinds of P2P software are obtained. Ethereal prints the data using text file format for analysis. After simple processing, the most useful information is obtained, such as the protocol type, the source IP address, the source port, the destination IP address, the destination port number, and DATA segment. DATA segment is the key object which we analyze; other parts of packets are only used for the later output and artificial analysis.
The following is a TXT format data packet outputted by Ethereal.

Figure 1. TXT format data packet outputted by Ethereal

```
No.  Time  Source  Destination  Protocol Info
 16 0.657392  219.133.248.3  128.59.19.185  TCP  38771 > 45234 [PSH, ACK]
Seq=0 Ack=0 Win=65493 Len=111
Frame 16 (165 bytes on wire, 165 bytes captured)
Ethernet II, Src: 128.59.16.1 (00:d0:06:26:9c:00), Dst: IntelCor_59:7e:f8
(00:13:20:59:7e:f8)
Internet Protocol, Src: 219.133.248.3 (219.133.248.3), Dst: 128.59.19.185
(128.59.19.185)
Transmission Control Protocol, Src Port: 38771 (38771), Dst Port: 45234 (45234), Seq: 0,
Ack: 0, Len: 111
Data (111 bytes)
0000  42 d8 8c d4 aa 94 cc 82 69 9a f7 e8 88 6e 99 f0   B.......i....n..
0010  c9 cd e1 12 a8 ac e3 1b dc 24 3f af 4f e0 b0 26   .........$?.O..&
0020  7b 0a a0 50 7d 7b c8 53 d2 b2 d6 09 9f c8 f9 dd   {..P}{.S........
0030  b5 fa 4a 19 0f 7d e1 1b 2d fc b5 42 de c8 97 c2   ..J..}..-..B....
0040  10 35 41 0e a5 72 f2 2e be 66 03 49 1e 49 0b 22   .5A..r...f.I.I."
0050  e4 9d e8 db cf d0 1a fe a4 91 94 1c ff ee bd d6   ................
0060  6e 1f 40 c2 42 63 94 5c e9 5f 90 99 f7 50 33      n.@.Bc.\._...P3
```

After simple processing, the protocol type, the source IP address, the source port, the destination IP address, the destination port number and DATA segment are obtained as following.

Figure 2. Data packet after simple processing

```
TCP 219.133.248.3 128.59.19.185 38771 45234 > 42 d8 8c d4 aa 94 cc 82 69 9a f7 e8 88
6e 99 f0 c9 cd e1 12 a8 ac e3 1b dc 24 3f af 4f e0 b0 26 7b 0a a0 50 7d 7b c8 53 d2 b2 d6
09 9f c8 f9 dd b5 fa 4a 19 0f 7d e1 1b 2d fc b5 42 de c8 97 c2 10 35 41 0e a5 72 f2 2e be
66 03 49 1e 49 0b 22 e4 9d e8 db cf d0 1a fe a4 91 94 1c ff ee bd d6 6e 1f 40 c2 42 63 94
5c e9 5f 90 99 f7 50 33
```

TCP in figure2 represents the transport layer protocol is TCP. ‘219.133.248.3’ is the source IP address. ‘128.59.19.185’ is the destination IP address. ‘38771’ and ‘45234’ are source port and destination port respectively. ‘>’ is the start symbol of data segment, and subsequence characters are content of data segment.

2.3.2 Data storage using Key Tree structure

The payload of each data packets is inserted into data structure, so that Key Tree is constructed step by step. Here, each element of the Key Tree is one byte of the payload. Therefore the highest frequency bytes can be identified, and the position of it can also be located. The main steps are as following.

(1) Reading the data payload of data packets.

(2) Partitioning the data payload into elements. (Each element of Key Tree is one byte.)

(3) Inserting every element into Key Tree.

(4) Using method provided by Key Tree to obtain the frequency table in descending order.

(5) Getting the high frequency application signatures.

It is very simple and easy to implementation using Key Tree. There is some sample data which is chosen from real-data in table 1. 'No.1, No.2… No.8' represents identifier of each data packet. '0, 1…13' in first row represents the index of bytes in each packet. Because different packets have different length, the short packet will be filled with '*' to up to enough length.

Table 1. Sample data which is chosen from real data

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
No.1	03	3f	02	3b	8a	55	97	05	12	1a	6c	7d	46	97
No.2	69	36	02	97	c9	35	9c	75	9c	58	ea	4c	42	68
No.3	57	8d	02	7f	a4	0c	ee	cb	fa	a4	ac	19	a1	8b
No.4	69	38	02	d1	95	b1	ea	24	38	cf	92	8f	67	18
No.5	57	93	02	b5	24	1a	0d	18	9f	e9	fd	54	67	7b
No.6	5e	64	A6	c8	ce	75	1b	b1	63	de	a5	fb	*	*
No.7	69	3a	02	97	71	8e	68	d7	95	8f	2a	24	0c	7d
No.8	97	ad	1f	7f	26	ca	eb	3e	a0	75	25	96	87	6b

In fact, the count of corresponding bytes in the same location can be operated in column direction. So, we establish a Key Tree for each column, and store each Key Tree into a link list (each node of the list is a Key Tree). The values in the column and the identifiers of each value are stored in the nodes of Key Tree. In order to examine the packet identifier of each value, the identifier is also stored in a link list. After all Key Trees are created, the frequent 1-sequences can be found by using the statistical function in Key Tree. But it can only find the application signatures whose size is only one byte. It could not organize effective memory structure to discovery long frequent sequence.

2.3.3 Frequent 2-Sequence mining

The next step is the focus of frequent 2-itemsets generation. It is the union process for frequent 1-itemset produced in last step. It should be noticed whether the two items are in same position. If they are in same position, they will not be united. The union is not only for bytes, but also for location and identifier of packets. In last step, the frequent and the identifier of values are recorded, while the locations are not recorded. The location can be obtained according to the index of the list. So it is not convenient for farther using. We should design a simple storage structure for recording the useful information outputted by Key Trees and the location information in the list. The key problem is the synchronization for the value and the location. If there are any mistakes, the subsequence process can not continue. It is not only a process for union and filter, but also a process for storage structure reorganization. It will provide a storage structure for long sequence mining. We design a class called longSignal to store frequent 2-sequences. There are three lists to store variables. Information list is used to store the 2 bytes value of 2-sequence (the application signature). Position list is used to store the identifier number of data packets. Situation list is used to store the location of the application signature of data packets. The elements in situation list must be corresponding with the elements in information list.

2.3.4 Frequent n- Sequence mining

The mining process will become easy as a result of the forming of fixed format after frequent 2-sequence. As long as we pay attention to the synchronization of the location and the byte value, we can use the union and selection functions in new storage format to implement maximum sequence mining.

In the frequent n-sequence creating process, union is different from traditional sequence mining algorithm. This algorithm only connects data in the same data packet. If the location of data has overlap, the value of data in the same numerical position must be same. So there are three limitation factor which we must consider, the identify number of data packets for L_{n-1}, the position of L_{n-1} and the value of L_{n-1}. This makes the algorithm is complicated than the traditional sequence mining method. The synchronization between the byte value and location transform to the synchronization information list and situation list. If there is an operation in either information list or situation list, the other will also do the operation meanwhile.

The main methods are as following.

- void inOrder() // Sorting the information list and situation list according to the situation value.
- boolean isSameSituation()// Judging whether they are appearing in the same packet. If the byte value in the same situation is different, they are impossible to be found in the same data packet.
- boolean isValueable(int total, float level)// Judging whether it is worthy of reserving according to the min-support. Here, total is the overall number of data packets, and level is the minimum support threshold.
- void union()//Determining whether two sets can be merged by using isSameSituation (), then union two longSignal, so that the item with same situation will be merged.
- boolean equals()//If the three lists of two longSignal are same, then they are equal.

The completeness of the algorithm is tested on virtual data. The role of the algorithm in the practical applications is also tested based on real data test.

3 Experiments

3.1 Testing on simulate data

In order to get results on different conditions, we test the algorithm using simulate data. The input simulate data is as following.

> 01 23 03 23 44
> 23 23 03 33
> 01 23 93 84 77
> 01 23 03 43 44
> 66 32 03 45 67
> 55 44 93 57 83

> 44 02 03 04 42
It contains the DATA segments of 7 data packets. We can find that the '01 23 03 44' occurs in 0^{th}, 1^{st}, 2^{nd}, 4^{th} bytes of 1^{st} and 4^{th} packets. '01 23' and '23 03' appears three times separately. The highest frequent character is '03' which occurs five times.
If the min-support is 25%, the output is '[01, 23, 03, 44] [0, 1, 2, 4] [1, 4]'. Here, [01, 23, 03, 44] represents the application signature; [0, 1, 2, 4] represents the location of the application signature; [1, 4] represents the number of packets which contain the application signature.

If the min-support is up to 40%, the output has two sequences.
[01, 23] [0, 1] [1, 3, 4];
[23, 03] [1, 2] [1, 2, 4].
First sequence represents that '01' and '23' occur in 0^{th} and 1^{st} byte of 1^{st}, 3^{rd} and 4^{th} packets. Second sequence represents that '23' and '03' occur in 1^{st} and 2^{nd} byte of 1^{st}, 2^{nd} and 4^{th} packets.

If the min-support is up to 50%, the output has two 1-sequences.
[23] [1] [1, 2, 3, 4]
[03] [2] [1, 2, 4, 5, 7]
First sequence represents that '23' occurs in 1^{st} byte of 1^{st}, 2^{nd}, 3^{rd} and 4^{th} packets. Second sequence represents that '03' occurs in 2nd byte of 1^{st}, 2^{nd}, 4^{th}, 5^{th} and 7^{th} packets.

3.2 Testing on real data

We also test the algorithm on real data which is scratched from network as skype application is login. It is known that the 3^{rd} byte in DATA segments of lot of packets is '02'. The algorithm finds the sequence. [02] [2] [7, 8, 55, 56, 59, 60, 65, 66, 140, 142, 143, 144, 145, 146, 148, 149, 150, 152, 153, 156, 158, 159, 176, 177, 178, 179, 180, 181, 298, 299, 300, 301]. '[02]' represents that '02' is the application signature. '[2]' represents that the location of application is 2^{nd}. In factually, it is the 3^{rd} byte of the DATA segment, but the index of line is from 0. '[7, 8, 55, 56, 59, 60, 65, 66, 140, 142, 143, 144, 145, 146, 148, 149, 150, 152, 153, 156, 158, 159, 176, 177, 178, 179, 180, 181, 298, 299, 300, 301]' represents the identifier of packets which value 3^{rd} byte is 02. Compared with real data, we can find that theresult is correct. For example, the DATA segment of 7^{th} packet is '03 3f 02 3b 8a 55 97 05 12 1a 6c 7d 46 97 51 5c 24 91 6f b3 eb 9e a3'.

4 Conclusions

This paper presents an application signature discovery method. The algorithm can be said to be a variant of Apriori algorithm. The design of the new algorithm is not completely in accordance with the mode of Apriori algorithm, because it must focus on the location information. The kernel idea of this algorithm is from the

Apriori algorithm, but the coding process is completely independent. We determine the completeness of the algorithm through test on virtual data. Based on real data test, we can see the role of the algorithm in practical applications. Although the algorithm does not yet have a good ability to discover low frequent signatures, it provides a high frequent signature discovery method. The other hand, since the algorithm adopts idea of Apriori, therefore it is still unable to avoid some of the drawback of Apriori algorithm. The efficiency of discovery exceed long sequence is low. Since P2P application signatures are not too long, the low efficiency of such extreme circumstances can be tolerated. Meanwhile, we can also set up a limitation of the length of signatures. So that, we could get segments of exceed long sequences quickly, without a long time waiting.

Acknowledgement

This research is partially sponsored by Beijing Municipal Education Committee (No.KM200610005012) and Beijing Excellent Talent Special Foundation (No.20042D0501504).

References

[1] Karagiannis T, Broido A, Faloutsos M, Claffy Kc. Transport Layer Identification of P2P Traffic. In: proceedings of the 4th ACM SIGCOMM conference on Internet measurement, Taormina, Sicily, Italy, ACM Press, 2004;121–134

[2] Sen S, Spatscheck O, Wang DM. Accurate. Scalable In-Network Identification of P2P Traffic Using Application Signatures. In: proceedings of the 13th international conference on World Wide Web, New York, NY, USA, ACM Press, 2004; 512–521

[3] Agrawal R, Srikant R. Mining Sequential Patterns. In: Yu PS, Chen ALP, eds; Proc. of the 11th Int'l Conf. on Data Engineering, ICDE'95; Taipei: IEEE Computer Society, 1995; 3-14

[4] Srikant R, Agrawal R. Mining Sequential Patterns: Generalizations and Performance Improvements. In: Apers PMG, Bouzeghoub M, Gardarin G; Proc. of the 5th Int'l Conf. on Extending Database Technology: Advances in Database Technology; London: Springer-Verlag, 1996; 3-17