BitMiner: Bits Mining in Internet Traffic Classification

Zhenlong Yuan[*‡], Yibo Xue[†‡] and Mihaela van der Schaar[‖]
[*]Department of Automation, Tsinghua University, Beijing, China
[‖]Department of Electrical Engineering, UCLA, Los Angeles, CA, USA
[†]Tsinghua National Lab for Information Science and Technology, Beijing, China
[‡]Research Institute of Information Technology, Tsinghua University, Beijing, China
yuanzl11@mails.tsinghua.edu.cn, yiboxue@tsinghua.edu.cn, mihaela@ee.ucla.edu

ABSTRACT

Traditionally, signatures used for traffic classification are constructed at the byte-level. However, as more and more data-transfer formats of network protocols and applications are encoded at the bit-level, byte-level signatures are losing their effectiveness in traffic classification. In this poster, we creatively construct bit-level signatures by associating the bit-values with their bit-positions in each traffic flow. Furthermore, we present BitMiner, an automated traffic mining tool that can mine application signatures at the most fine-grained bit-level granularity. Our preliminary test on popular peer-to-peer (P2P) applications, e.g. *Skype*, *Google Hangouts*, *PPTV*, *eMule*, *Xunlei* and *QQDownload*, reveals that although they all have no byte-level signatures, there are significant bit-level signatures hidden in their traffic.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network management*

Keywords

Traffic classification, bit-level signatures, bits mining

1. INTRODUCTION

Signature based traffic classification has been playing an important role in a broad range of network operations and security management, such as quality-of-service control and intrusion detection. However, due to the increasing number of network applications and their frequent updates, it is becoming more challenging for operators to keep track of the signatures.

To address this challenge, a number of existing solutions have focused on automatically extracting signatures at the byte-level [4, 5], which first divide packet payloads into groups of consecutive bytes and then analyze to get the possible signatures. However, those solutions have two major limitations. Firstly, they are unable to discover signatures at the more fine-grained bit-level granularity. Note that previous work [1,2] have revealed that bit-level characteristics (group of 4 bits, less than 1 byte) are of great importance in identifying a few P2P applications. Secondly, they confine signatures to groups of consecutive bytes and thus are hard to discover the signatures that consist of inconsecutive bytes (e.g. 1 byte) in packet payloads. In this poster, we propose the novel bit-level signatures, and present an automated traffic mining tool (BitMiner) that can mine signatures at the most fine-grained bit-level granularity.

2. BITMINER

In this poster, we have two observations. The first is that an application signature should be robust enough to support per-flow identification due to the prevalence of asymmetric routing. For this reason, a favorable application signature should be one of the most frequent patterns in captured traffic after running an application for plenty of times. Therefore, our goal can turn into mining the most frequent patterns[1] in the application traffic. The second is that the bit-value of a bit-position in a flow often determines the bit-values of other bit-positions in this flow. Therefore, we are motivated to associate all the bit-values with their bit-positions in a flow for frequent pattern mining (signature mining).

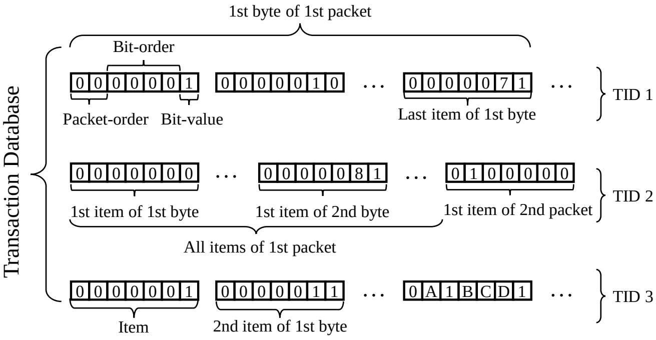

Figure 1: Format Traffic Flows to Transactions

As shown in Figure 1, we can take a bit-value with its position in a flow as an *item* and take all the bit-values with their individual positions in this flow as a *transaction*. Notice that we use only two hexadecimal characters to represent an *item*'s packet-order in a flow because application signatures are generally required to achieve early identification in practical use and the first 256 (0x00∼0xFF) packets of a flow are

[1] From here, we start using some *terms* in *Data Mining*.

Transaction Database		
TID	Original Items	Sorted Items
01	I4, I5, I7, I8	I7, I4, I5, I8
02	I4, I6, I7, I9	I7, I4, I9, I6
03	I1, I2, I3, I5	I1, I2, I3, I5
04	I1, I2, I3, I7	I7, I1, I2, I3
05	I1, I2, I3, I5, I9	I1, I2, I3, I5, I9
06	I4, I5, I7, I8	I7, I4, I5, I8
07	I1, I2, I3, I9	I1, I2, I3, I9
08	I4, I6, I7, I9	I7, I4, I9, I6
09	I1, I2, I3, I6	I1, I2, I3, I6
10	I4, I5, I7, I8	I7, I4, I5, I8

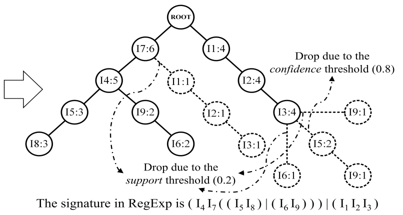
ROOT
I7:6
I1:4
I4:5
I1:1
I2:4
I5:3
I9:2
I2:1
I3:4
I9:1
I8:3
I6:2
I3:1
I5:2
I6:1
I9:1
Drop due to the *confidence* threshold (0.8)
Drop due to the *support* threshold (0.2)
The signature in RegExp is ($I_4 I_7$ (($I_5 I_8$) | ($I_6 I_9$))) | ($I_1 I_2 I_3$)

Figure 2: An Example of How *BitMiner* Works

<table>
<tr><th>Applications</th><th>Application Signatures</th><th>Support (Recall)</th></tr>
<tr><td>Skype</td><td>^(002_0x02)+(002_0_0 & 002_4_1 & 002_5_1 & 002_6_0 & 002_7_1)*$</td><td>100.00%</td></tr>
<tr><td>Xunlei (Thunder)</td><td>^(001_0_0 & 003_0_0 & 003_1_1 & 003_2_0)*$</td><td>100.00%</td></tr>
<tr><td>eMule</td><td>^(000_0_0 & 000_4_0 & 001_0_0 & 001_4_0 & 000_6_0 & 001_1_0 & 001_5_0) | (000_0_0 & 000_4_0 & 001_0_0 & 001_4_0 & 000_6_1 & 001_1_1 & 001_5_1)*$</td><td>100.00%</td></tr>
<tr><td>Google Hangouts</td><td>^(000_0_1 & 000_1_0 & 001_1_1 & 001_3_0)*$</td><td>100.00%</td></tr>
<tr><td>PPTV (PPLive)</td><td>^(007_0_0 & 007_1_0 & 007_2_0 & 007_3_0 & 008_0x00 & 009_0_0 & 009_1_0 & 009_2_0 & 009_3_0 & 009_6_0 & 00A_0_0 & 00A_1_0 & 00A_2_0 & 00A_3_0)*$</td><td>100.00%</td></tr>
<tr><td>QQDownload</td><td>^(000_1_1 & 000_2_1 & 000_5_1 & 000_7_0 & 001_0_0 & 001_1_0 & 002_0_0 & 002_1_0 & 002_7_0 & 003_0_0 & 004_0_0 & 005_0_0 & 007_5_0 & 009_0_0 & 00A_1_0)*$</td><td>100.00%</td></tr>
</table>

Table 1: The Generated Bit-level Signatures

sufficient enough. Similarly, we use four hexadecimal characters to represent one *item*'s bit-order in a packet payload because the MTU of an IP packet over Ethernet networks is 1500-byte where 1 byte has 8 bit-orders.

BitMiner consists of two major parts: *Bit-table* and *Miner-tree*. Figure 2 shows an example of how BitMiner works. *Bit-table* is a hash table used for hashing and storing all the *items* read from a *transaction database*. In this process, *Bit-table* will read the transaction database twice. For the first time, *Bit-table* will only count the *support* of every *item*. For the second time, *Bit-table* will remove the items whose *support* is below the initially set *support threshold* and sort the remaining *items* in every *transaction* by their *supports* (maximum to minimum). After that, all the sorted *transactions* will be entered into *Miner-tree* as a new *transaction database*.

Miner-tree is a prefix tree of the new *transactions*, which takes idea from the FP-tree [3] but is different. Note that there are probably multiple tasks running within an application and thus the signature could be a regular expression. Considering a *transaction* (flow) can only belong to one of the tasks, all the *transactions* are divided into multiple clusters to represent different tasks. Since the *items* in each *transaction* have been sorted by their *supports*, it is extremely fast to construct the *Miner-tree*. Moreover, as a signature does not need to be so long as the number of *items* in a *transaction*, a *level* parameter is set to restrict the maximum depth (e.g. 800) of the *Miner-tree*, which can save memory consumption and make the construction much faster.

After constructing the *Miner-tree*, there will be a pruning process controlled by two thresholds: *minimum support* and *minimum confidence*. Firstly, the *support* (defined as the proportion of *transactions* in a node from the whole *transaction database*) will be checked for every single node. After that, the *confidence* (defined as the proportion of *transactions* in all the child-nodes of a node from the node itself) will be checked for every parent node. In this way, it can be determined whether a branch should be removed or a parent node should stop splitting. Finally, the branches of the pruned *Miner-tree* are the target signature.

3. EVALUATION

BitMiner has been tested on the UDP traffic of six popular P2P applications (they all use UDP protocol for transmitting a significant amount of traffic which brings serious challenges to network management). As shown in Table 1, every signature is generated by BitMiner within a few seconds. The signatures are written in a customized way. The "(p)" represents a pattern (p) matching within one packet's payload, the "$\wedge(p)$" represents this matched packet is the first packet of a flow, the "$(p)\$$" represents this matched packet is the last packet of a flow, the "$(p)+$" represents this matched packet appears one or more times in succession within a flow, the "$(p)*$" represents this matched packet appears zero or more times in succession within a flow, the "`002_0x02`" represents the `third` byte value of a packet's payload is `0x02`, the "`002_4_1`" represents the `fifth` bit value of the `third` byte is `1`, the "*p&p*" represents two patterns matching with one packet's payload simultaneously and the "$(p)|(p)$" represents either one matched packet appears within a flow. For instance, the `third` byte values of the first one or more packets of a Skype flow are always `0x02` while `five` bit values of the `third` bytes of all the other packets are fixed. Specially, we also examine the other bits adjacent to the mined ones, such as the '*second*, *third* and *fourth*' bits of the `third` bytes of Skype flows and the '*fourth*, *fifth*, *sixth*, *seventh* and *eighth*' bits of the `fourth` bytes of Thunder[2] flows. The results show that those bit-values are completely random (i.e. uniformly distributed).

Also as shown in Table 1, the *support* represents the proportion of flows matched with the mined signature, which is equivalent to the *recall* in traffic classification. In addition, a longer signature generally means a better *precision*. For example, if we check the first 10 packets of a Thunder flow, the signature used for matching is totally 40 bits long, which may be robust enough to get a high precision in real-world traffic classification.

4. REFERENCES

[1] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing skype traffic: when randomness plays with you. In *ACM SIGCOMM*, 2007.

[2] A. Finamore, M. Mellia, M. Meo, and D. Rossi. Kiss: stochastic packet inspection classifier for udp traffic. *IEEE/ACM Transactions on Networking*, 2010.

[3] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD*, 2000.

[4] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker. Unexpected means of protocol inference. In *ACM SIGCOMM IMC*, 2006.

[5] Z. Zhang, Z. Zhang, P. P. Lee, Y. Liu, and G. Xie. Proword: an unsupervised approach to protocol feature word extraction. In *IEEE INFOCOM*, 2014.

[2] The most popular P2P file sharing software in China