Expert Systems with Applications 37 (2010) 8850–8860

Contents lists available at ScienceDirect

Expert Systems with Applications

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

Mining frequent patterns from network flows for monitoring network

Xin Li, Zhi-Hong Deng *

Peking University, Key Laboratory of Machine Perception (Ministry of Education), School of Electronic Engineering and Computer Science, Room 2318, Science Buildings 2, 100871 Beijing, China

ARTICLE INFO

Keywords:
Network monitoring
Anomaly detection
Frequent pattern mining
Sliding window

ABSTRACT

Because of the varying and dynamic characteristics of network traffic, such as fast transfer, huge volume, shot-lived, inestimable and infinite, it is a serious challenge for network administrators to monitor network traffic in real time and judge whether the whole network works well. Currently, most of the existing techniques in this area are based on signature training, learning or matching, which may be too complicated to satisfy timely requirements. Other statistical methods including sampling, hashing or counting are all approximate methods and compute an incomplete set of results. Since the main objective of network monitoring is to discover and understand the active events that happen frequently and may influence or even ruin the total network. So in the paper we aim to use the technique of frequent pattern mining to find out these events. We first design a sliding window model to make sure the mining result novel and integrated; then, under the consideration of the distribution and fluidity of network flows, we develop a powerful class of algorithms that contains vertical re-mining algorithm, multi-pattern re-mining algorithm, fast multi-pattern capturing algorithm and fast multi-pattern capturing supplement algorithm to deal with a series of problems when applying frequent pattern mining algorithm in network traffic analysis. Finally, we develop a monitoring system to evaluate our algorithms on real traces collected from the campus network of Peking University. The results show that some given algorithms are effective enough and our system can definitely identify a lot of potentially very valuable information in time which greatly help network administrators to understand regular applications and detect network anomalies. So the research in this paper not only provides a new application area for frequent pattern mining, but also provides a new technique for network monitoring.

1. Introduction

As the Internet keeps moving, everyone would have more and more dependence on the network. A secure, stable and efficient network is necessary for today's society. So, monitoring the network globally and timely is critically important for network administrators to make sure the whole network running regularly and working availably. Generally speaking, network administrators care more about the frequent happened events that may threaten or influence the efficiency of network than individual or sporadic activities. The main objectives in network monitoring can be summarized as two aspects: understanding traffic features which appear especially most frequently; and detecting outburst network anomalies.

Given a large-scale campus network or enterprise network, there always exist a lot of network flows which have similar traffic features. Flows are usually considered to be sequences of packets with a 5-tuple of common values (protocol, source and destination IP addresses and port numbers), and ending after a fixed timeout interval when no packets are observed. For example, the packets send (or received) by a DNS server in a local network always have this DNS server's IP address and port 53 as source or destination address and port. To understand this kind of traffic feature is equal to understand the comparatively constant frequent pattern in network flows. So by observing these network activities, network administrators can be more conscious of how these regular applications work and may find a better way to reduce unnecessary traffic load for configuration optimization and capacity planning. Anomaly detection is other important task in network monitoring, outburst malicious flows may flood the whole network and severely degrade network performance. Most of the anomalies correspond to some special patterns in the distributing of infected flow. For example, Distributed Denial of Service (DDoS) is one of dangerous Internet attacks, in which a multitude of compromised hosts attack a single target, thereby causing denial of service for users of the targeted system. So all packets infected by a particular DDoS attack have the same destination IP, the DDoS attack could correspond to a ⟨multi-source IPs, destination IP⟩ pattern. No matter understanding traffic load or anomaly detection, they both are essentially finding frequently happened features in data packets, defined by source/destination IP, protocols, source/destination

* Corresponding author. Tel.: +86 10 62755592; fax: +86 10 62754911.
E-mail address: zhdeng@cis.pku.edu.cn (Z.-H. Deng).

0957-4174/$ - see front matter Crown Copyright © 2010 Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.06.012

port, or other information extracted from the header of packets. In other words, it means mining frequent patterns from network flows. In this paper we try to use the technique of frequent pattern mining to implement a network monitoring system.

However, the varying and dynamic characteristics of network traffic including fast transfer, huge volume, shot-lived, inestimable and infinite, make it very difficult to implement efficient monitoring in real time. So we first design a sliding window model for data processing to make sure the mining result novel and integrated; then, under the consideration of the distribution and fluidity of network flows, we develop a powerful class of algorithms, which contains sliding window re-mining algorithm, multi-pattern re-mining algorithm, multi-pattern fast mining algorithm and multi-pattern fast mending mining algorithm to deal with problems when applying frequent pattern mining algorithm in network traffic analysis. Finally, we develop a monitoring system to evaluate our algorithms on real traces collected from the campus network of Peking University. The results show that our system can definitely identify a lot of potentially very valuable information in time, which greatly help network administrators to understand regular applications and detect network anomalies. So the research in this paper not only provides a new application area for frequent pattern mining, but also provides a new technique for network monitoring.

2. Background

2.1. Intrusion detection

Intrusion detection is a general problem in Network Security era. Traditional methods for intrusion detection are based on extensive knowledge of attack signatures provided by human experts. The significant limitations of signature-based methods are that they cannot detect novel attacks; in addition, recent research indicates that signature-based network intrusion detection systems are quickly becoming ineffective at identifying malicious traffic (Crandall, Su, Wu, & Chong, 2005; Song, Locasto, Stavrou, Keromytis, & Stolfo, 2007). These limitations have led to an increasing interest research based upon data mining, which generally fall into one of two categories: misuse detection and anomaly detection .

In misuse detection, each instance in a dataset is labelled as 'normal' or 'intrusive' and a learning algorithm is trained over the labelled data (Ghosh & Schwartzbard, 1999; Lippmann & Cunningham, 2000). Unlike signature-based method, models of misuse are created automatically and can be more sophisticated and precise than manually created signatures. However, their still cannot detect attacks whose instances have not yet been observed, and labelling data instances as normal or intrusive may require enormous time for many human experts.

Anomaly detection algorithms build models of normal behaviour and automatically detect any deviation from it (Denning, 1987; Javitz & Valdes, 1993). The major benefit of such algorithms is their ability to potentially detect unforeseen attacks. But the major limitation is a possible high false alarm rate. There are two major categories of anomaly detection techniques, namely supervised and unsupervised. Recently, there have been several efforts in designing supervised network-based anomaly detection algorithms, such as ADAM (Sushi, 2006), PHAD (Tandon & Chan, 2005). However, the efficiency of supervised anomaly detection methods depends heavily on the quality of the data used to train them. While most realistic datasets are dirty; that is, they contain a number of attacks or anomalous events (Cretu, Stavrou, Locasto, Stolfo, & Keromytis, 2008). Unsupervised anomaly detection approaches are based on statistical approaches, clustering, outlier detection schemes, etc (Eskin, Arnold, Prerau, Portnoy, & Stolfo, 2002; Staniford, Hoagland, & McAlerney, 2002) and may product lots of false alarms. So, even the popular anomaly-based approaches are also not perfect (Fogla & Lee, 2006; Taylor & Gates, 2006).

2.2. Frequent pattern mining

Given a transaction database DB and a minimum support threshold ξ, the problem of mining frequent patterns is to discover the complete set of patterns that have support greater than $\xi\% \times |DB|$, where a frequent pattern is a set of items that occur together in at least $\xi\%$ of DB, and the support of a frequent pattern is the times it occurs in DB.

Since mining frequent patterns was first introduced in Agrawal, Imielinski, and Swami (1993), it has emerged as a fundamental problem in data mining and plays an essential role in many important data mining tasks such as associations, sequential patterns, particle periodicity, classification, etc (Han, Pei, & Yin, 2000). This problem has been studied over ten years, and a lot of mature and popular algorithms have been proposed and well used in many application areas, including decision support, market strategy and financial forecast. Most of the previous proposed frequent patterns mining algorithms can be clustered into three groups: the Apriori-like method (Agrawal & Srikant, 1994), the FP-growth method (Han et al., 2000) and vertical method (Zaki, 2000; Zaki & Gouda, 2003).

The Apriori-like approach generates candidate patterns of length $(k+1)$ in the $(k+1)$th pass using frequent patterns of length k generated in the previous pass and counts the supports of these candidate patterns in the database. The idea of Apriori-like approach depends on an anti-monotone Apriori property (Agrawal & Srikant, 1994): all nonempty subset of a frequent pattern must also be frequent. A lot of studies, such as Zaki (2000) and Shenoy et al. (2000), adopt the Apriori-like approach. However, previous studies reveal that it is high expensive for Apriori-like approach to repeatedly scan the database and check a large set of candidates by pattern matching (Han et al., 2000).

The FP-growth algorithm has proved to be very efficient in mining frequent patterns. It achieves impressive efficiency by adopting a highly condensed data structure called frequent pattern tree to store databases and employing a partitioning-based, divide-and-conquer method to mining frequent patterns. Some studies, such as Liu, Lu, Xu, and Yu (2003), Han et al. (2000) and Han, Pei, Yin, and Mao (2004), adopt the FP-growth approach. The FP-growth approach wins an advantage over the Apriori-like approach by reducing search space and generating frequent patterns without candidate generation. However, the process of recursively constructing and mining conditional frequent pattern trees is complex (Woon, Ng, & Lim, 2004) and tends to need a large number of memories to store these temporal pattern trees.

Unlike the traditional horizontal transaction database format used in most Apriori-like algorithms, each item in a vertical database is associated with its corresponding Tid-list—the set of all transaction IDs where it appears. The advantage of vertical database format is that the counting of support for each frequent pattern can be obtained via Tid-list intersection, which avoids scanning a whole database. Tid-list is much simpler than complex hash or trees used in horizontal algorithms and also more efficient than them in counting supports of frequent patterns. Eclat (Zaki, 2000) and dEclat (Zaki & Gouda, 2003) are two representative vertical algorithms. Vertical mining methods have been shown to be very effective and usually outperform horizontal mining methods (Zaki & Gouda, 2003). So we will apply and improve this vertical mining method for network monitoring in this paper.

2.3. Stream mining

Recently, algorithms of mining frequent patterns in stream data become a hot topic in research area. Stream data are much difficult

to handle than traditional static database because the data are flowing continuously, temporally, inestimably and infinitely. Network flow is exactly one kind of stream data. The characteristic of fluidity makes those frequent pattern mining algorithms which modelled on static dataset unsuitable and inapplicable on stream data. Though, many different stream mining algorithms have proposed, most of them still rest on the theoretic study stage, lucking flexibility and feasibility in real application. Such as MG (Karp, Shenker, & Papadimitriou, 2003), Level-Blocks (Arasu & Manku, 2004), hCount (Jin, Qian, Sha, Yu, & Zhou, 2003), etc. They can only calculate frequent 1-pattern. But multi-pattern is more useful for analysis and decision. FP-stream (Giannella, Han, Pei, Yan, & Yu, 2003), In-Core Streaming (Jin & Agrawal, 2005), estDe (Chang & Lee, 2003) are algorithms mining frequent patterns in stream data, but they focus all their energy on the spatial limitation and hardly show consideration and take care to the temporal efficiency. In addition, many algorithms obtain an approximate set of frequent patterns, rather than exact ones.

3. Model and framework

In this section, we first introduce a sliding window model to handle network flow and give an overview of our network monitoring system based on this handling model.

Sliding windows are a typical method of flow control, especially for network data transfers. So we build a sliding window in fixed length, which means the size of this window is decided by a fixed length period of time, such as 10 min or 5 min. The sliding window should contain all the fresh data in network; so once a new piece of data appears, the sliding window should move forward to capture it. Because there are hundreds even thousands of new data generated in one second in network flows, to avoid updating sliding window too frequently, we use basic windows with equal length (such as one minute or thirty second) to divide a sliding window into several continuous partitions. So only when a new basic window comes, the sliding window will moves forward to capture it and drop the oldest one at the same time. See example in Fig. 1, a sliding window is composed of j basic windows; when a new basic window BW_{i+j} comes which contains all the freshest data in network flows, the sliding window moves from position 1 to position 2 to capture it and drop the overdue basic window BW_{i-1}.

Fig. 2 shows the framework of our network monitoring system. All packets in network are captured in real time, and flow information, such as source/destination IP, protocols, source/destination port, and other information, is extracted from each packet, all of which compose into a new transaction represented for a sequence of packets. Currently this work is implemented as NetFlow in Cisco routers. Then, these transactions are sent into a basic window, which caches all new incoming flow data. When it is time up for this incoming basic window full, it triggers three actions: (1) the system sliding window start to capture the data in the incoming basic window and drop the overdue data into cache that stores the latest outdated basic window; (2) the incoming basic window is cleared to collect new data from the beginning; (3) call frequent pattern mining algorithm on this updated sliding window to obtain

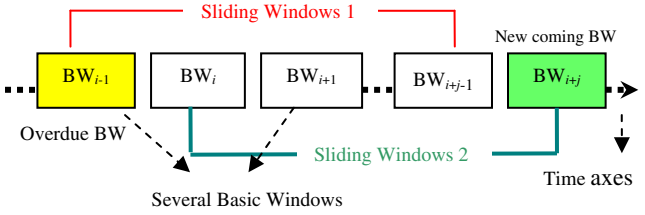

Fig. 1. The sliding window model.

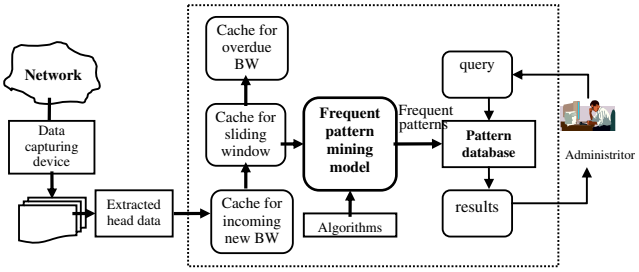

Fig. 2. System framework.

frequent patterns at this time. We store the mining results in database, so network administrators can directly examine all frequent events happened on the network easy and analyse them in real time. The efficiency and effectivity of the frequent pattern mining algorithm on sliding window is the core of our system. Whether the frequent pattern mining is applicable for network monitoring is all depends on the performance of the mining algorithm, so we focus all our attention on the discussion and development of a practicable algorithm in following sections in this paper.

4. Frequent pattern mining algorithm on network flows

In this section, we first give a review of the classic vertical mining algorithm and discuss the problem when using this algorithm directly on the network data. Then, we introduce a fast update algorithm which is the key point for frequent pattern mining being capable of online network monitoring. However, the situation of dynamically mining and maintaining the frequent patterns is complex, so we fully discuss all the incontrollable situations encountered by fast update algorithm and overcome them by exploiting and making best use of the peculiar characteristics of network flows.

4.1. Vertical frequent pattern mining algorithm

As discussed in Section 2 (B), vertical method (Zaki, 2000; Zaki & Gouda, 2003) is one classical frequent pattern mining methods. Since the infinite problem is turned into a finite problem by sliding window model, we directly use the algorithm of vertical method on the dataset within a sliding window. The detailed process is described in Table 1. In the beginning, the algorithm scans the whole dataset to filter out all frequent 1-patterns with their supports. Then, scan dataset again to record Tid-list for each frequent 1-pat-

Table 1
Vertical frequent pattern mining algorithm.

Input: dataset in a sliding window DB_s and the minimum support ξ.
Output: the complete set of frequent patterns
Method:

(1) Scan DB_s once to find $L[1] = \{frequent1\text{-}patterns\}$;
(2) Scan DB_s again to collect $TL[1] = \{the\ Tid\text{-}list\ of\ each\ frequent1 - pattern\ in L[1]\}$;
(3) **for** $(k = 2; L[k-1] \neq \varnothing; k{+}{+})$ **do begin**
(4) **for** all $p \in L[k-1]$ and $q \in L[k-1]$, where $p[1] = q[1], \ldots, p[k-2] = q[k-2], p[k-1] < q[k-1]$ **do begin**
(5) $c = p[1], p[2], \ldots, p[k-1], q[k-1]$; //Candidate k-pattern
(6) c.Tid-list = code-intersection(p.Tid-list, q.Tid-list); //linear
(7) **If** (c.support $\square |DB| \times \xi$) **then**
(8) $L[k] = L[k] \cup \{c\}$;
(9) $TL[k] = TL[k] \cup \{c.Tid - list\}$;
(10) **end if**
(11) **end for**
(12) Delete $TL[k-1]$; // No use of $TL[k-1]$ any more
(13) **end for**
(14) Answer = $L[1] \cup L[2] \cup \cdots \cup L[k]$.

tern. The Tid-list of a particular item contains all the IDs of transactions in which this item occurs, and the length of the Tid-list is its support. So when calculate the support of a 2-pattern, we can simply intersect the two Tid-lists of the corresponding 1-patterns composing that 2-pattern. Because the intersection contains the IDs of all transactions in which both the two 1-patterns occur. It is the same way to get all other frequent k-patterns ($k \geqslant 2$).

Using this vertical mining algorithm, we can directly get an integrated and accurate set of frequent patterns. The problem is that, whenever the sliding window moves forward to a new position, we have to call this algorithm to scan the whole dataset in current sliding window and mine them from beginning to update results. However, one move only changes a small part of the sliding window, most of the data within the window unchanged. It also means that one move of the sliding window only brings a small change of the frequent patterns. So calling this vertical mining algorithm every time (we called it vertical re-mining method in the following text) will do a lot of repeated scan and repeated mining work. Obviously, it is inefficient. In addition, if the length of a basic window is set very small which means the sliding window updates very frequently, or there are a very large number of transactions in the sliding window, the time spend by this re-mining algorithm may be too long to satisfy online monitor.

4.2. Fast update mining algorithm

Since most part of two neighbouring sliding windows is the same, we develop a fast update mining algorithm to avoid repeated scanning and mining between them. The fast update algorithm works based on an independent coding technique within each basic window.

When we use the traditional vertical mining algorithm to mine frequent patterns from a sliding window, remember that each transaction is numbered with a unique ID at first in that sliding window, denoted by ID_{SW}. And each frequent pattern keeps a Tid-list, which contains all the ID_{SW} in which this pattern occurs. In our fast update algorithm, we first number transactions based on basic window instead of sliding window, denoted by ID_{BW}. See example in Table 2. Suppose that there are thirty transactions in a sliding window and the sliding window is equably divided by five basic windows. So these transactions are numbered from 1 to 30 by ID_{SW}, but numbered from 1 to 6 by ID_{BW} within the corresponding basic window. So in our fast update algorithm, each frequent pattern keeps n Tid_{BW}-lists, where n is the number of basic windows in a sliding window. The advantage of using ID_{BW} is that when a new basic window comes, all overlapped basic windows in the current sliding window just keep the original Tid_{BW}-lists without repeated computation. It only needs to collect the Tid_{BW}-list from the new incoming basic window, and updates the support by totalizing the length of all overlapped Tid_{BW}-lists and this new Tid_{BW}-list. So this independent coding greatly reduces the mining time.

Table 2
Transctions in A sliding window with five basic windows.

ID_{SW}	ID_{BW}	Transaction	ID_{SW}	ID_{BW}	Transaction
1	1	A B	16	4	A C
2	2	A	17	5	C
3	3	A B	18	6	B C
4	4	B	19	1	C
5	5	B C	20	2	C
6	6	B C	21	3	A C
7	1	A B C	22	4	A
8	2	A	23	5	A B C
9	3	A C	24	6	A B C
10	4	A	25	1	A B
11	5	A	26	2	A
12	6	A B C	27	3	A B
13	1	A B C	28	4	A B
14	2	A	29	5	A C
15	3	B C	30	6	B C

Fig. 3 shows the mining structure based on the data in Table 2. When the system calls the fast update mining algorithm for the first time, it runs the same processes with vertical mining in Table 1 except using the independent Tid_{BW}-lists to calculate the support instead of the uniform Tid_{SW}-list. Fig. 3(a) shows the complete mining process during the first mining. Every pattern keeps five Tid_{BW}-lists there, and only the Tid_{BW}-lists belongs to the same basic window intersect together to obtain a new Tid_{BW}-list. When all the frequent patterns are mined out, we delete all the Tid_{BW}-lists but record their lengths in the pattern tree, showed in Fig. 3(b). The support of each frequent pattern can be easy calculated by totalizing the record of each basic window together. From then on, whenever the sliding window moves to a new position, our fast update mining algorithm only need to scan and calculate the Tid_{BW}-list for each frequent pattern in that new basic window, add the length of the new Tid_{BW}-list into pattern tree for every frequent pattern, and delete the oldest record at the same time. Then, output the frequent patterns with there updated support in pattern tree and all the infrequent ones can be directly cut down from the pattern tree.

The fast update mining algorithm gets the frequent patterns with their support in the current sliding window by just dealing with a small part of data within the newest basic window, so it improves the performance quite a lot compared with vertical re-mining algorithm. However, it only updates all frequent patterns that already exist in pattern tree. Besides these patterns, there are many other patterns, which were not frequent before but may become frequent later. To use the fast update mining algorithm, we must find means to capture these new emerging frequent patterns. The situation is complex, so we divide this problem into two sub-problems: capturing new frequent 1-patterns and capturing new frequent multi-patterns, and settle them separately in the following two sections.

4.3. Capturing new frequent 1-patterns by candidate queue

The core advantage of fast update mining algorithm is that it avoids the repeated scanning and mining of the data in all overlapped basic windows. However, fast update mining algorithm fails to get the support of a frequent 1-pattern that is not in the pattern tree as not frequent in the last sliding window, but appears frequent in the new incoming basic window. See example of 1-patterns in Fig. 4 continuing the example in Fig. 3, patterns A, B, C are all frequent before, so they have detailed record within each

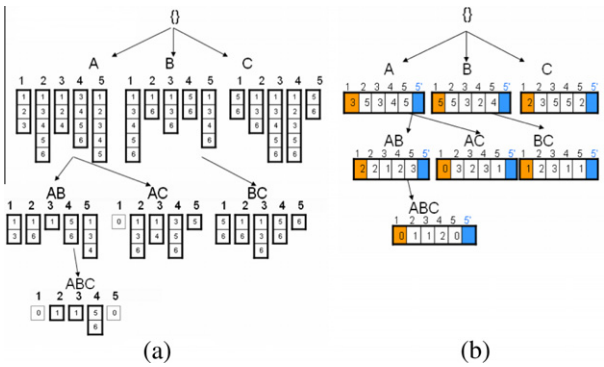

Fig. 3. The first running of fast update algorithm and the pattern tree.

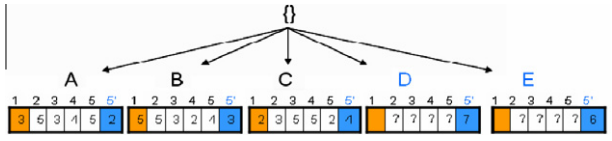

Fig. 4. Example of new emerging frequent 1-patterns.

basic window in the pattern tree. However, patterns D and E are not frequent before, so their relative information is missing.

To avoid repeated scanning still, here we develop a method called candidate queue to capture all the new emerging frequent 1-patterns. Once there is one such 1-pattern appearing at the first time, we put it in a queue as a candidate pattern, and record its corresponding Tid_{BW}-list collected from that new incoming basic window. The next time another basic window comes, we check if it still keeps frequent locally in these two basic windows together. If not, we delete it from the candidate queue, else, record its new Tid_{BW}-list collected from the second basic window. The same step is performed until when there are n record of Tid_{BW}-lists for that 1-pattern in the queue (n is the number of basic windows in a sliding window). At this time, we can make sure it is a new frequent 1-pattern. So we allocate a new node in pattern tree to denote it and delete it from candidate queue. Since the pattern is placed in the pattern tree, we can use the fast update mining algorithm to calculate its support quickly from then on.

Our candidate queue is worked based on the actual distribution of network flow. Usually, there are two types of 1-patterns transferred in network flows: (1) common practice, appearing with a steadily frequency; (2) sudden events, appearing very frequently at some point, and maybe continue from then on or disappear at once. For the first type, if it is frequent, which means that network administrators may get interested with it, our pattern tree has already captured it. For the second type, if it is a meaningful sudden events, which means it suddenly becomes frequent at some point and continues for a while, our candidate queue will successfully capture it finally. In addition, once a 1-pattern appears frequent at some point, even re-scanning of the original data is almost useless and time wasting, because it hardly appears before that point.

The problem of capturing new frequent 1-patterns has been solved by candidate queue without repeated scanning. Here we introduce a multi-pattern re-mining algorithm to evade the problem of capturing new frequent multi-patterns which fast update mining algorithm failed to mine. Simply speaking, the multi-pattern contains three steps:

(1) Scan the new incoming basic window BW to collect 1-patterns that exist in pattern tree, or exist in candidate queue, or appear frequent in BW with their Tid_{BW}-list.
(2) Use fast update mining algorithm to calculate only frequent 1-patterns in pattern tree.
(3) Update candidate queue: (a) add new 1-patterns into it, (b) delete unsatisfied ones, (c) for the ones that are frequent globally, delete them as well and allocate new nodes in pattern tree to denote them.
(4) Calculate the support of frequent $(k+1)$-patterns using the Tid-lists of frequent k-pattern (be the same with steps in Table 1 from (3) to (14))

The multi-pattern re-mining algorithm uses two steps: (2) and (3) to obtain all the frequent 1-patterns instead of re-scanning all data in sliding window. All frequent multi-patterns have to be mined from the beginning. So the pattern tree used here only contains one level of nodes which express all the frequent 1-patterns, and each node has to save the corresponding n Tid_{BW}-lists which will be used to calculate the support of multi-patterns.

4.4. Fast frequent multi-pattern capturing algorithm

The multi-pattern re-mining algorithm is not effective enough from the algorithm point of view. Because it cannot avoid the repeated mining of multi-patterns, where fast update mining algorithm has already deal with it. So to use the fast update mining algorithm furthest, we still need to solve the problem of capturing new emerging multi-patterns which may be potentially frequent.

We first discuss the situations that may produce new frequent multi-patterns. Theoretically speaking, if a multi-pattern becomes frequent from infrequent, it must belong to one of the following situations.

- Combined by all new emerging frequent 1-patterns.
- Combined by new and old frequent 1-patterns which still keep frequent currently.
- Combined by all old frequent 1-patterns already in pattern tree and keeping frequent currently.

For the first situation, it is easy to calculate the support, because we have saved all the Tid_{BW}-lists of every new frequent 1-pattern in candidate queue, we can directly intersect them to obtain the Tid_{BW}-lists of this new multi-pattern as well as its support. However, for the last two situations, we lost the detailed information of old frequent patterns to calculate the new one. Because we cannot store all patterns' Tid_{BW}-lists in the pattern tree, especially for the multi-patterns' Tid_{BW}-lists in the pattern tree which is space-consuming and may use up all the memory. So it is very difficult to capture this part of new multi-patterns, especially for the third situation that combined by all old frequent 1-patterns. Even we store the complete Tid_{BW}-lists for every old frequent 1-pattern in the pattern tree, we have to do the intersection from the first level of the pattern tree to the last level again to estimate if a new leaf node may generate which denotes a new frequent multi-pattern. So it is time-consuming and does a lot of repeated calculations for all the multi-patterns already known frequent in the pattern tree.

Since the steps used to capture all new frequent multi-patterns are so complicated and may destroy the performance in the mass, we turn to observe the distribution and characteristics of network flows again and find that maybe we can ignore a part of new multi-patterns produced under some situations discussed before. We find that usually a server or a host that appears very active in the network almost always is combined within a fixed pattern. In other word, if the server or host is active, the corresponding pattern is active, vice versa. So we abstract this observation and give the following supposition:

If the sub-pattern of an infrequent multi-pattern is frequent all the while, there is hardly any chance for this multi-pattern to become frequent.

See example in Fig. 5(a), if 1-patterns A, B, C and 2-pattern AB are all frequent last time in the pattern tree, but 2-pattern BC is

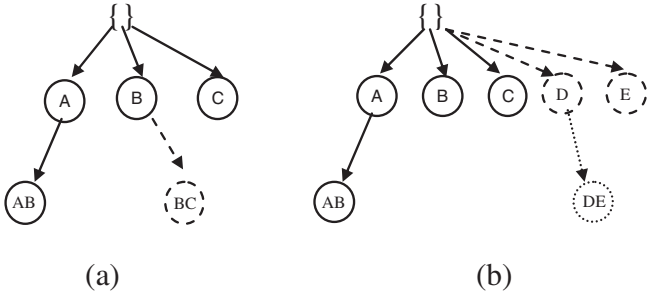

Fig. 5. Example of new emerging frequent multi-pattern.

infrequent, we can say that B and C have little relationship. So at this mining time, if A, B, C are still frequent, then AB have a big chance to keep frequent, but BC has little probability to become frequent which can be omitted.

Based on this supposition, we can omit the new multi-patterns combined by all old frequent 1-patterns. But for the new multi-patterns combined by both new and old 1-patterns, see example in Fig. 5(b). Here, patterns A, B, C and AB are old frequent patterns, and keep frequent currently. Patterns D and E are new frequent 1-patterns, and we need to check whether they will compose a new 2-pattern DE. But how about multi-pattern AD, AE, ADE, BD...? They all combined by new and old frequent 1-patterns. By the conclusion of our observation, an active host is apt to be combined within a fixed pattern. So it seems that we can omit this kind of new multi-pattern as well. However, sometimes maybe a active network server is changed to perform another task, which leads to a new multi-pattern composed by old and new elements. Hence, we cannot determine whether or not to omit this part of new emerging multi-patterns without a thorough estimation in practice. So we develop two algorithms here: fast frequent multi-pattern capturing algorithm and fast frequent multi-pattern capturing supplement algorithm. The former one only consider new multi-patterns combined by new frequent 1-patterns besides using fast update mining algorithm and candidate queue method; the latter does all the same process but makes a supplement by considering the new multi-patterns combined by new and old frequent 1-patterns.

Table 3 shows the fast frequent multi-pattern capturing algorithm in detail. This algorithm only scans a small dataset in the new incoming basic window; fully uses the fast update mining algorithm to update the support values of all known patterns in the pattern tree; then, use the candidate queue to capture new frequent 1-patterns and generate new frequent multi-patterns by the intersection of these new frequent 1-patterns' Tid_{BW}-lists, which is the key technique of vertical mining method.

The supplement form of fast frequent multi-pattern capturing algorithm performs all the steps described in Table 3 in same order, except increasing a sub-procedure before returning the final results. The increased sub-procedure is used to mine frequent multi-patterns composed by both old and new added frequent 1-patterns in pattern tree. So the pattern tree has to store n Tid_{BW}-lists in each node that denotes a frequent 1-pattern (n is the number of basic windows in a sliding window).

Table 4 summarizes and compares the four frequent pattern mining algorithms discussed in this section which are all designed specially for network flow mining.

Table 3
Fast frequent multi-pattern capturing algorithm.

Input: dataset in the new incoming basic window DB_b, the minimum support ξ, the pattern tree P_{tree}, the candidate queue Q_{can}.
Output: the complete set of frequent patterns
Method:

(1) Scan DB_b once to find $L[1] = \{1-$ patterns inP_{tree} or Q_{can}, or new local frequent ones in $DB_b\}$;
(2) Scan DB_b again to collect $TL[1] = \{$the Tid_{BW}-list of each 1-pattern in $L[1]\}$;
(3) Call fast update mining algorithm on P_{tree} to update it;
(4) Update Q_{can}
  (a) Obtain new local frequent ones in $L[1]$;
  (b) Delete unsatisfied ones;
  (c) Collect new frequent 1-patterns N[1], and delete them from Q_{can}.
(5) Make nodes in P_{tree} for N[1] = { new frequent 1-patterns}; //Next, generate new multi-patterns composed by N[1]
(6) **for** $(k = 2; N[k-1] \neq \varnothing; k{+}{+})$ **do begin**
(7) **for** all $p \in N[k-1]$ and $q \in N[k-1]$, where $p[1] = q[1], \ldots, p[k-2] = q[k-2], p[k-1] < q[k-1]$ **do begin**
(8) $c = p[1], p[2], \ldots, p[k-1], q[k-1]$; //Candidate k-pattern
(9) $c.Tid_{BW}$-lists = intersection($p.Tid_{BW}$-lists, $q.Tid_{BW}$-lists);
(10) **If** (c.support $\geqslant |DB| \times \xi$) **then**
(11) $N[k] = N[k] \cup \{c\}$;
(12) $NL[k] = NL[k] \cup \{c.$Tid-list$\}$;
(13) Construct a new node for c in P_{tree} as the children of p;
(14) **end if**
(15) **end for**
(16) Delete $NL[k-1]$; // No use of $NL[k-1]$ any more
(17) **end for**
(18) Answer = all nodes in P_{tree}.

5. Experiments

5.1. Experimental setup

To evaluate our proposed frequent pattern mining algorithms, we gather network traces from the campus network of Peking University. Peking University is one of the largest universities in China. Currently there are 2 Gbps access links between the campus network and CERNET (China Education and Research Networks). The campus backbone's bandwidth is 10 Gbps. The average number of online computers in campus is about 25,000. We deployed a network traffic collecting system, which receives NetFlow data from a border switch. The average amount of compressed NetFlow data per day is about 13 GB. Our experiments are performed on a Pentium(R) 4 CPU 3.00 GHz computer with 512 memory, running Microsoft Windows XP. All algorithms are coded in C++ and compiled by Microsoft visual studio.NET 2003.

When implementing the monitoring system in real time, we set the size of the sliding window 5 min and the size of basic window 1 min. So every minute the system performs the mining algorithm and updates the results one time.

To compare the performance of our proposed four frequent pattern mining algorithm, we randomly choose network data on the day of 05, March, 2008. In the next section, we show the running instances of our algorithms during a time period from 16:00 to 16:15. Because of the limitation of pages, we cannot show results on all data, and even this small time period contains 15 times update of the sliding window and brings on 15 times mining. The other reason is that we find the scale of NetFlow data in campus network follows rules along with different time period in a day. For example, the network is inactive in the early morning or later night, but very active in other time periods. So the experimental results enumerated later well indicate our algorithms' capability and expansibility. Table 5 shows the scale of the sliding window from 16:00 to 16:15 on 05, March, 2008; it presents that there are more than 3,600,000 transactions to be mined each time. By setting the minimum support to 0.1%, we can mine about 2000 frequent patterns in every 5 min. Practically speaking, that is enough for network administrators' timely monitoring, so the minimum support is small enough.

5.2. Algorithm evaluation

5.2.1. Vertical re-mining algorithm

The introduced vertical re-mining algorithm can be summarized into three main steps: (a) Scan DB once to find frequent 1-pattern; (b) Scan DB again to collect Tid-lists; (c) Mine all frequent multi-patterns circularly. Fig. 6 shows the total mining time as well as time consumed on each single step. Obviously, the time consumed on two scans of the dataset takes the majority proportion of the total time since the big scale of the dataset. Correspondingly, the key mining step is very fast. As a whole, this algorithm satisfies the Network Monitor every minute one time in this given instance but not effective enough.

Table 4
A summary of frequent pattern mining algorithm on network flow.

Algorithm	Data scan	Space	Accuracy	Integrity	Efficiency
Vertical re-mining	Repeated scan SW	No temporary structure	Accurate result	Integrated	Re-scanning and re-mining
Multi-pattern re-mining	Only scan new BW	Dynamically maintain frequent 1-patterns with their Tid_{BW}-list	Accurate result	Integrated	Re-mining
Fast multi-pattern capturing	Only scan new BW	Dynamically maintain a pattern tree	Accurate result	Missing frequent multi-patterns combined by old or old and new 1-patterns	Avoid re-scanning And re-mining
Fast multi-pattern capturing supplement	Only scan new BW	Dynamically maintain a pattern tree with frequent 1-patterns' Tid_{BW}-list	Accurate result	Missing frequent multi-patterns combined by old frequent 1-patterns	Avoid re-scanning and re-mining

Table 5
The number of transactions in each sliding window during the time period from 16:00 to 16:15.

The end moment of SW	Number of transactions	The end moment of SW	Number of transactions	The end moment of SW	Number of transactions
16:01	3,640,511	16:06	3,711,287	16:11	3,661,745
16:02	3,661,418	16:07	3,698,435	16:12	3,663,615
16:03	3,682,325	16:08	3,685,583	16:13	3,665,485
16:04	3,703,232	16:09	3,651,824	16:14	3,667,355
16:05	3,724,139	16:10	3,659,875	16:15	3,669,229

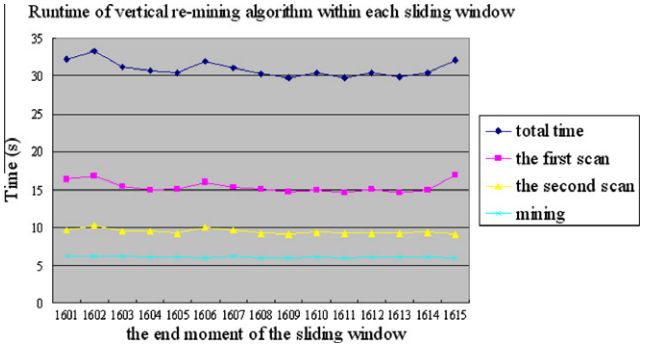

Fig. 6. Runtime of vertical re-mining algorithm.

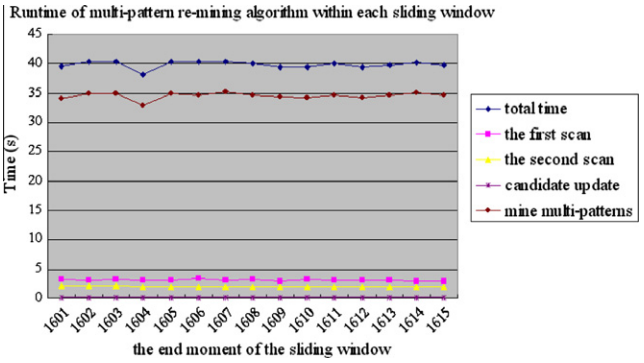

Fig. 8. Runtime of multi-pattern re-mining algorithm.

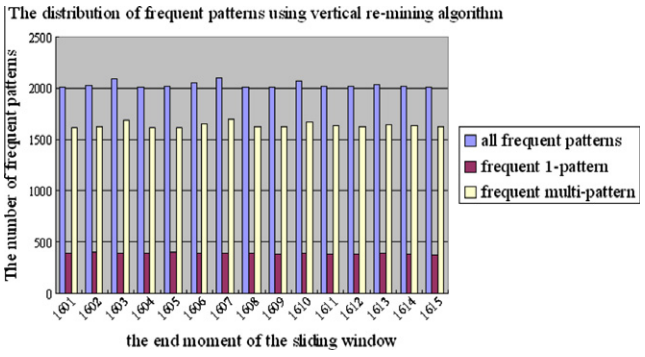

Fig. 7. The distribution of frequent patterns using vertical re-mining algorithm.

Fig. 7 shows the mining results, about 2000 frequent patterns every time, by setting the minimum support to 0.1%. Obviously, the proportion of multi-patterns is almost 3 times bigger than 1-patterns' in each mining. It validates that using frequent pattern mining for network monitoring is meaningful and can surely find out lots of long patterns, which summarizes and indicates some potential useful information. And by using the vertical re-mining algorithm, we can accurately mine the complete set of all frequent patterns.

5.2.2. Multi-pattern re-mining algorithm

The multi-pattern re-mining algorithm first scans the new incoming basic window twice, then uses candidate queue to capture new frequent 1-patterns, and mines k-patterns by using the intersections of (k-1)-patterns' Tid-lists circularly ($k \geqslant 2$). Since the size of a basic window only takes 1/5 of the sliding window, the time consumed by the two scans in Fig. 8 is much smaller than the same steps in Fig. 6. However, the multi-pattern mining step is so slow even multi-times slower than the same step in Fig. 6. The reason is that, when applying vertical re-mining algorithm, we use a skill of generating all 2-patterns during the second scanning of the original dataset (Zaki & Gouda, 2003), which avoids the time-consuming on the heavy intersection of frequent 1-patterns' Tid-lists. However, by applying multi-pattern re-mining algorithm, we cannot use this skill as this algorithm does not scan the total dataset within the sliding window. As a whole, the multi-pattern re-mining algorithm is worse than vertical re-mining algorithm, and ineffective.

Remarkably, the step of candidate queue's update hardly spends any time. In Fig. 9, we find that the proportion of new frequent 1-patterns and the ones becoming infrequent is so inappre-

ciable compared with others. It validates that the distribution of NetFlow data is steady, and any change of it usually happened locally and inappreciably most of time. So a frequent pattern tends to keep frequent for a while. This discovery indicates that our fast update mining algorithm may just fit this situation and work well.

5.2.3. Fast frequent multi-pattern capturing algorithm

Fig. 10 shows the detailed time-consuming by fast frequent multi-pattern capturing algorithm. As expected, the fast update algorithm performs fast and steady, only takes one second and a little. The time spending on two scans of the basic window becomes the most time-consuming steps instead. As a whole, this algorithm performs well and effective.

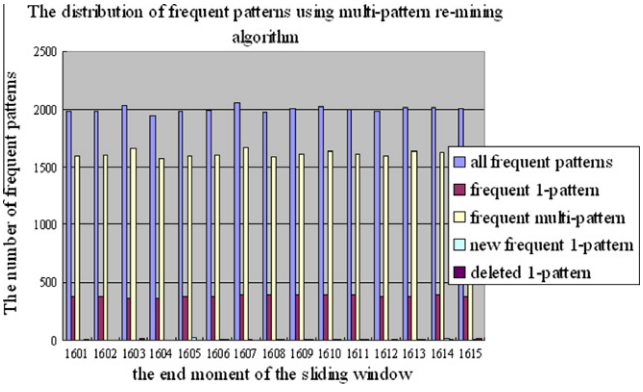

Fig. 9. The distribution of frequent patterns using multi-pattern re-mining algorithm.

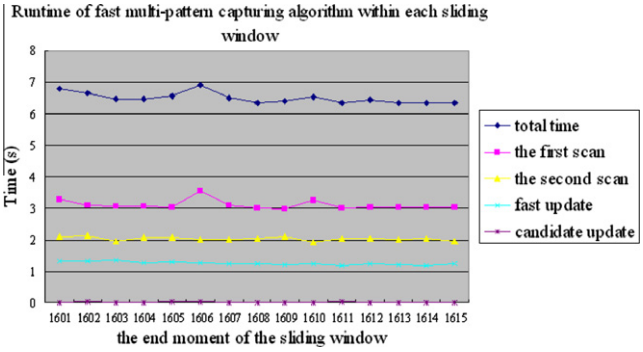

Fig. 10. Runtime of fast multi-pattern capturing algorithm.

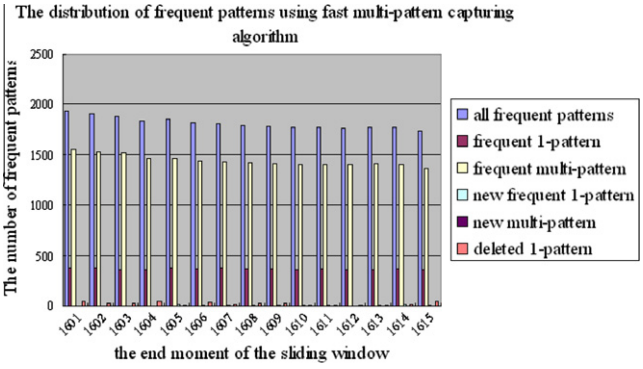

Fig. 11. The distribution of frequent patterns using fast multi-pattern capturing algorithm.

Fig. 11 shows the distribution of frequent patterns using fast multi-pattern capturing algorithm. It validates again that any change of NetFlow data usually happened locally and inappreciably. So our supposition in Section 4. G comes into existence in our experiment.

5.2.4. Fast frequent multi-pattern capturing supplement algorithm

Fig. 12 shows the detailed runtime of fast frequent multi-pattern capturing supplement algorithm. It much likes the trend of curves in Fig. 10, except including another curve which indicates the step of capturing the new multi-patterns combined by new and old frequent 1-patterns. This time spent by this supplement step is uneven, since it has to check all possible candidate multi-patterns by interaction of Tid-lists. So this supplement algorithm

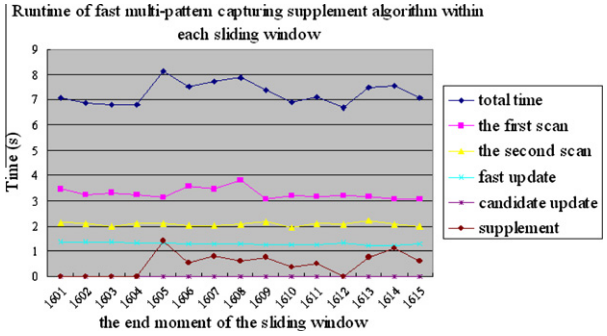

Fig. 12. Runtime of fast multi-pattern capturing supplement algorithm.

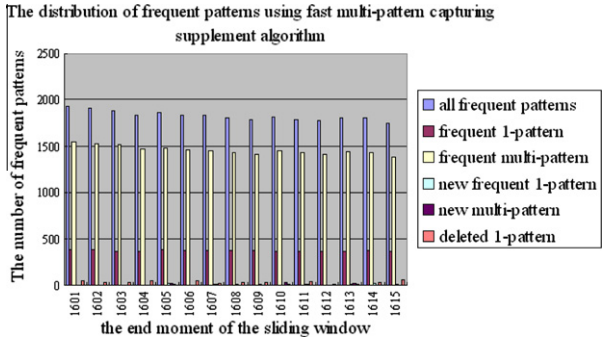

Fig. 13. The distribution of frequent patterns using fast multi-pattern capturing supplement algorithm.

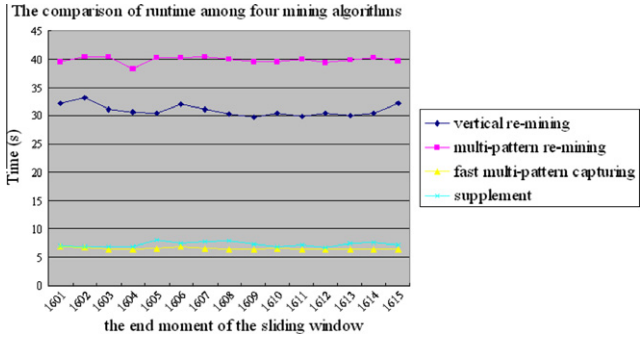

Fig. 14. Comparison of runtime among four frequent pattern mining algorithms.

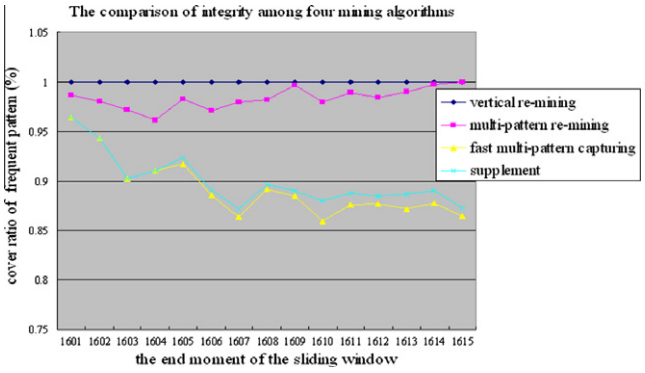

Fig. 15. Comparison of mining results among four mining algorithms.

spends a little more time than fast frequent multi-pattern capturing algorithm as a whole.

Fig. 13 shows the detailed distribution of frequent patterns. Even the last supplement step may do a lot of jobs, there still little new multi-patterns can be found out. It validates our supposition again.

5.2.5. Comparison

Figs. 14 and 15 show the total comparison of our proposed four frequent pattern mining algorithms.

- The vertical re-mining algorithm can find the complete set of frequent patterns, where time-consuming is on a middle level.
- The multi-pattern re-mining algorithm can also find the complete set of frequent patterns but with a delay, because of the candidate queue's temporal cache operation. However, the efficiency is worst.
- The two fast multi-pattern capturing algorithms speed up the total mining process. They are both much faster than the other algorithms. From the side of mining result, they cover more than 85% of the complete set of frequent patterns. The ones they lost are combined by old 1-patterns, which are very sensitive to the value of the minimum support. So these lost multi-patterns are the least frequent ones in the complete set.

So the two fast multi-pattern capturing algorithms will not lost the important frequent patterns. And we can make use of them by choosing a proper minimum support.

5.3. Experience with frequent pattern mining

After mining, all these frequent patterns are inserted into database for future query and analysis. This section presents highlights of our experience with frequent pattern mining, some examples are enumerated, which are based on the data within the sliding window from 16:25 to 16:30 on 05, March, 2008. When the minimum support threshold is 0.1%, we get over 2000 frequent patterns from over 3,000,000 flows whose supports are no less than 3000.

In the first query, for reducing the number of targeted (frequent) patterns, we select the patterns which include more than 4 attributes from the pattern set, because long patterns contain more information than short ones. So we get 71 long patterns in this special instance. Then, we use the maximal pattern (Gouda & Zaki, 2001) to summary it and shrinks to 40 patterns from 71 patterns. These 40 patterns are list below. For privacy and security, we anonymize IP addresses in the patterns using partial prefix preservation.

srcIP	desIP	prot	srcPort	desPort	octets	packets	support	length	time
	192.168.146.24	17		30610	105	1	48810	5	2008-03-05-1625
192.168.146.24		17	30610		92	2	42713	5	2008-03-05-1625
	192.168.146.24	17	30701	30610		1	36066	5	2008-03-05-1625
		17	30701	30610	105	1	36050	5	2008-03-05-1625
	192.168.146.24	17	30701	30610	105	1	36049	6	2008-03-05-1625
	192.168.146.24	17	30701		105	1	36049	5	2008-03-05-1625
192.168.146.24		17		30701	92	2	34619	5	2008-03-05-1625
192.168.146.24		17	30610	30701		2	31617	5	2008-03-05-1625
		17	30610	30701	92	2	31546	5	2008-03-05-1625
192.168.146.24		17	30610	30701	92	2	31542	6	2008-03-05-1625
	202.112.7.13	17		7777	322	7	17690	5	2008-03-05-1625
192.168.146.12	222.29.154.19	17		834		1	15144	5	2008-03-05-1625
192.168.146.15	222.29.154.19	17		834		1	14216	5	2008-03-05-1625
	222.29.154.19	17		834	120	1	13609	5	2008-03-05-1625
202.112.7.13		17	7777		322	7	12147	5	2008-03-05-1625
	192.168.146.24	17		30610	210	2	10481	5	2008-03-05-1625
		17	6110	6112	46	1	9092	5	2008-03-05-1625
222.29.154.19		17	834		60	1	7117	5	2008-03-05-1625
192.168.146.12	222.29.154.19	17		834	120	1	7092	6	2008-03-05-1625
192.168.161.238		6		443	120	2	6823	5	2008-03-05-1625
192.168.146.88		17	974		50	1	6775	5	2008-03-05-1625
192.168.146.15	222.29.154.19	17		834	120	1	6517	6	2008-03-05-1625
	222.29.154.19	17		834	124	1	6067	5	2008-03-05-1625
	192.168.146.24	17		30609	328	1	5673	5	2008-03-05-1625
	192.168.129.60	6		80	46	1	5667	5	2008-03-05-1625
	222.29.154.19	17		834	116	1	5455	5	2008-03-05-1625
222.216.28.125		6	6000	3080		1	5339	5	2008-03-05-1625
222.216.28.125		6	6000	3080	46	1	5333	6	2008-03-05-1625
192.168.146.24		17	30610		184	4	5094	5	2008-03-05-1625
222.199.232.170		17		0	46	1	5080	5	2008-03-05-1625
192.168.146.88		17	974	30601		1	4933	5	2008-03-05-1625
192.168.146.88		17	974	30601	50	1	4898	6	2008-03-05-1625
192.168.146.24		17	30609		92	2	4795	5	2008-03-05-1625
192.168.161.238		6		443	60	1	4497	5	2008-03-05-1625
	192.168.146.88	17		974	71	1	4243	5	2008-03-05-1625
		1	0	0	56	1	4145	5	2008-03-05-1625
202.112.7.13		17	7777		276	6	4144	5	2008-03-05-1625
59.66.141.76		17	6110	6112	46	1	3354	6	2008-03-05-1625
	192.168.146.24	17	30701	30609		1	3681	5	2008-03-05-1625
222.29.154.19	192.168.146.12	17	834		60	1	3643	6	2008-03-05-1625

5.4. Understand traffic features

(a) Case One – a kind of P2P file sharing applications
There exist some patterns below, which include "192.168.146.24" as srcIP or desIP using UDP port 30610 in the reduced 40 patterns. Based on known facts from the operators of the campus network, we know that the host of "192.168.146.24" is an index server of Maze, and the UDP port 30610 is default configuration about communication port. The index server store information about all files available on peer nodes, queries are resolved by index server. Because Maze is a large-scale P2P file sharing system in China, a lot of search queries make the patterns with these specific address and port significant. From the symmetric feature of normal request-reply traffic, we identify that these patterns stand for the exchange behaviour of file index in Maze. Furthermore, in other time period, last one hour, last one day, these patterns all exist, so this network feature is regular.

srcIP	desIP	prot	srcPort	desPort	octets	packets	support	length	time
	192.168.146.24	17		30610	105	1	48810	5	2008-03-05-1625
192.168.146.24		17	30610		92	2	42713	5	2008-03-05-1625
192.168.146.24		17		30701	92	2	34619	5	2008-03-05-1625
	192.168.146.24	17	30701		105	1	36049	5	2008-03-05-1625
	192.168.146.24	17	30701	30610	105	1	36049	6	2008-03-05-1625
192.168.146.24		17	30610	30701		2	31617	5	2008-03-05-1625
	192.168.146.24	17	30701	30610		1	36066	5	2008-03-05-1625
		17	30701	30610	105	1	36050	5	2008-03-05-1625
		17	30610	30701	92	2	31546	5	2008-03-05-1625
192.168.146.24		17	30610	30701	92	2	31542	6	2008-03-05-1625
	192.168.146.24	17		30610	210	2	10481	5	2008-03-05-1625
192.168.146.24		17	30609		92	2	4795	5	2008-03-05-1625
	192.168.146.24	17		30609	328	1	5673	5	2008-03-05-1625

(b) Case Two – Campus Logon Client's heat-beating
In Peking University, network users use a desktop logon client for logining into a charging system, the logining details are for usage charging. For indicating current network status of connectivity, this client will send a heat-beating message to the host of "202.112.7.13" in fixed time interval. So the following patterns give the evidence of clients' heat-beating, which is routine and normal.

srcIP	desIP	prot	srcPort	desPort	octets	packets	support	length	time
	202.112.7.13	17		7777	322	7	17690	5	2008-03-05-1625
202.112.7.13		17	7777		322	7	12147	5	2008-03-05-1625
202.112.7.13		17	7777		276	6	4144	5	2008-03-05-1625

5.4.1. Detect network anomaly

(a) Case One – Port Scan flows

srcIP	desIP	prot	srcPort	desPort	octets	packets	support	length	time
59.66.141.76		17	6110	6112	46	1	5534	6	2008-03-05-1625

In the pattern above, there exists an IP address of "59.66.141.76", which is outside of the address blocks of Peking University. From the patterns of last 20 min, no pattern which contains "59.66.141.76" as desIP exists and the size and number of packet is definitely same. So we conclude that this pattern stands for a Port Scan focusing on unused port. And similar analysis is suitable for the patterns below exploiting TCP and ICMP separately.

srcIP	desIP	prot	srcPort	desPort	octets	packets	support	length	time
222.216.28.125		6	6000	3080		1	5339	5	2008-03-05-1625
222.216.28.125		6	6000	3080	46	1	5333	6	2008-03-05-1625
222.199.252.170		17		0	46	1	5080	5	2008-03-05-1625

(b) Case Two – Flash or mis-configuration flows

srcIP	desIP	prot	srcPort	desPort	octets	packets	support	length	time
192.168.146.12	222.29.154.19	17		834		1	15144	5	2008-03-05-1625
192.168.146.15	222.29.154.19	17		834		1	14216	5	2008-03-05-1625
	222.29.154.19	17		834	120	1	13509	5	2008-03-05-1625
222.29.154.19		17	834		60	1	7117	5	2008-03-05-1625
192.168.146.12	222.29.154.19	17		834	120	1	7092	6	2008-03-05-1625
192.168.146.15	222.29.154.19	17		834	120	1	6517	6	2008-03-05-1625
	222.29.154.19	17		834	124	1	6067	5	2008-03-05-1625
	222.29.154.19	17		834	116	1	5455	5	2008-03-05-1625
222.29.154.19	192.168.146.12	17	834		60	1	3643	6	2008-03-05-1625

In the patterns above, there exists one IP address of "222.29.154.19" which has suspicious UDP connections with Maze servers ("192.168.146.12" and "192.168.146.15") by using a fixed UDP port 834. This behaviour just appears in last 10 min. So after the help of network operators, we identify that the Maze client in the host of "222.29.154.19" was mis-configured.

6. Conclusions

In this paper, we propose the problem of applying frequent pattern mining in the field of network monitoring. We give four algorithms: vertical re-mining algorithm, multi-pattern re-mining algorithm, fast frequent multi-pattern capturing algorithm, and fast frequent multi-pattern capturing supplement algorithm to mine frequent patterns from NetFlow data. Then, we develop a monitoring system based on the campus network of Peking University to evaluate our algorithms on real data. The results show that: (1) our algorithms are effective enough to be used on real time network monitoring, especially the algorithms of fast frequent multi-pattern capturing algorithm and its supplement algorithm which dynamically produce and maintain the frequent patterns; (2) our system can definitely find out a lot of potentially very valuable information in time which greatly enhances the ability of campus network monitoring. So the research in this paper is very valuable. In future, we plan to improve this work by: applying algorithms of closed-pattern mining, top-k pattern mining, or maximal mining to condense the mount of frequent patterns and make the result tidier and easier to understand; exploiting a system interface to automatically reveal the patterns with which the administrator may be interested; designing a self-learning model that can identify normal and anomaly actions automatically.

Acknowledgements

We would like to acknowledge the support from the National High Technology Research and Development Program (863 program in china: 2009AA01Z136) and the National Natural Science Foundation of China (90812001).

References

Agrawal, R., & Srikant, R. (1994). Fast algorithm for mining association rules. In *Proceedings of 1994 international conference on very large data bases (VLDB'94)* (pp. 487–499).

Agrawal, R., Imielinski, T. & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of 1993 ACM SIGMOD international conference on management of data (SIGMOD'93)* (pp. 207–216).

Arasu, A., & Manku, G.S. (2004). Approximate counts and quantiles over sliding windows. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems* (pp. 286–296).

Chang, J., & Lee, W.S. (2003). Finding recent frequent itemsets adaptively over online data streams. In *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 487–492). Washington, USA: ACM Press.

Crandall, J. R., Su, Z., Wu, S. F. & Chong, F. T. (2005). On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits. In *ACM conference on computer and communications security, Alexandria, VA*.

Cretu, G. F., Stavrou, A., Locasto, M. E., Stolfo, Salvatore J. & Keromytis, Angelos D. (2008). Casting out demons: Sanitizing training data for anomaly sensors. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy* (pp. 81–95).

Denning, D. E. (1987). An intrusion-detection model. *IEEE Transactions on Software Engineering, SE-13*, 222–232.

Eskin, E., Arnold, A., Prerau, M., Portnoy, L., & Stolfo, S. (2002). A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. *Data Mining for Security Applications*.

Fogla, P., & Lee, W. (2006). Evading network anomaly detection systems: Formal reasoning and practical techniques. In *Proceedings of the 13th ACM conference on computer and communications security (CCS)* (pp. 59–68).

Ghosh, A.K., & Schwartzbard, A. (1999). A study in using neural networks for anomaly and misuse detection. In *Proceedings of the eighth USENIX security symposium Washington, DC* (pp. 141–151).

Giannella, C., Han, J., Pei, J., Yan, X., & Yu, P. S. (2003). Mining frequent patterns in data streams at multiple time granularities. *Generation Data Mining*, 191–212.

Gouda, K., & Zaki, M. J. (2001). Efficiently mining maximal frequent itemsets. In *Proceedings of the first IEEE international conference on data mining* (pp. 163–170).

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. In *Proceedings of 2000 ACM SIGMOD international conference on management of data (SIGMOD'00)* (pp. 1–12).

Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery, 8*, 53–87.

Javitz, H. S., & Valdes, A. (1993). *The nides statistical component: Description and justification*. Technical report. Computer Science Laboratory, SRI International.

Jin, R. M., & Agrawal, G. (2005). An algorithm for in-core frequent itemset mining on streaming data. In *Fifth IEEE international conference on data mining (ICDM)* (pp. 210-217).

Jin, C. Q., Qian, W. N., Sha, C. F., Yu, J. X., & Zhou, A. Y. (2003). Dynamically maintaining frequent items over a data stream. In *Proceedings of the 12th ACM conference on information and knowledge management (CIKM)*.

Karp, R. M., Shenker, S., & Papadimitriou, C. H. (2003). A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems, 28*(1), 51–55.

Lippmann, R. P., & Cunningham, R. K. (2000). Improving intrusion detection performance using keyword selection and neural networks. *Computer Networks, 34*, 597–603 [(Amsterdam, Netherlands: 1999)].

Liu, G., Lu, H., Xu, Y., & Yu, J.X. (2003). Ascending frequency ordered prefix-tree: Efficient mining of frequent patterns. In *Proceedings of 2003 international conference on database systems for advanced applications (DASFAA'03)* (pp. 65–72).

Shenoy, P., Haritsa, J. R., Sundarshan, S., Bhalotia, G., Bawa, M. & Shah, D. (2000). Turbo-charging vertical mining of large databases. In *Proceedings of 2000 ACM SIGMOD international conference on management of data (SIGMOD'00)* (pp. 22–33).

Song, Y., Locasto, M. E., Stavrou, A., Keromytis, A. D. & Stolfo, S. J. (2007). On the infeasibility of modeling polymorphic shellcode. In *ACM computer and communications security conference (CCS)*.

Staniford, S., Hoagland, J. A., & McAlerney, J. M. (2002). Practical automated detection of stealthy portscans. *Journal of Computer Security, 10*, 105–136.

Sushi, J. (2006). *Integration of audit data analysis and mining techniques into aide*. AFRL-IF-RS-TR-2006-250. Final Technical Report, July.

Tandon, G., & Chan, P. (2005). Learning useful system call attributes for anomaly detection. In *Proceedings of 18th international FLAIRS conference* (pp. 405-410).

Taylor, C., & Gates, C. (2006). Challenging the anomaly detection paradigm: A provocative discussion. In *Proceedings of the 15th new security paradigms workshop (NSPW), September* (pp. 21–29).

Woon, Y. K., Ng, W. K., & Lim, E. P. (2004). A support-ordered trie for fast frequent pattern discovery. *IEEE Transactions on Knowledge and Data Engineering, 16*(7), 875–879.

Zaki, M. J. (2000). Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering, 12*(3), 372–390.

Zaki, M. J., & Gouda, K. (2003). Fast vertical mining using diffsets. In *Proceedings of 2003 international conference on knowledge discovery and data mining (SIGKDD'03)* (pp. 326–335).