Available online at www.sciencedirect.com
ScienceDirect
Information Sciences 178 (2008) 694–713
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

Privacy preserving data mining of sequential patterns for network traffic data ☆

Seung-Woo Kim [a], Sanghyun Park [a,*], Jung-Im Won [b], Sang-Wook Kim [b]

[a] *Department of Computer Science, Yonsei University, 134 Sinchon-dong, Seodaemun-gu, 120-749 Seoul, Republic of Korea*
[b] *College of Information and Communications, Hanyang University, Republic of Korea*

Received 2 November 2006; received in revised form 3 August 2007; accepted 25 August 2007

Abstract

As the total amount of traffic data in networks has been growing at an alarming rate, there is currently a substantial body of research that attempts to mine traffic data with the purpose of obtaining useful information. For instance, there are some investigations into the detection of Internet worms and intrusions by discovering abnormal traffic patterns. However, since network traffic data contain information about the Internet usage patterns of users, network users' privacy may be compromised during the mining process. In this paper, we propose an efficient and practical method that preserves privacy during sequential pattern mining on network traffic data. In order to discover frequent sequential patterns without violating privacy, our method uses the *N*-repository server model, which operates as a single mining server and the *retention replacement* technique, which changes the answer to a query probabilistically. In addition, our method accelerates the overall mining process by maintaining the *meta tables* in each site so as to determine quickly whether candidate patterns have ever occurred in the site or not. Extensive experiments with real-world network traffic data revealed the correctness and the efficiency of the proposed method.

Keywords: Data mining; Sequential pattern; Network traffic; Privacy

1. Introduction

The number of computers connected to the Internet and exchanging data via the Internet have dramatically increased, owing to the rapid advance of network technology. Recently, a new kind of data mining has appeared in which researchers extract useful knowledge from network traffic data that are automatically gathered by a remote server [6,12,15,19,27]. Identifying patterns of network intrusions and differentiating anomalous network activity from normal network traffic data are typical examples.

☆ This work was partially supported by the Korea Research Foundation Grant funded by the Korean Government (KRF-2005-041-D00651) and the ITRC support program supervised by the IITA (IITA-2005-C1090-0502-0009).
* Corresponding author. Tel.: +82 2 2123 5714; fax: +82 2 365 2579.
E-mail addresses: kimsw@cs.yonsei.ac.kr (S.-W. Kim), sanghyun@cs.yonsei.ac.kr (S. Park), jiwon@hanyang.ac.kr (J.-I. Won), wook@hanyang.ac.kr (S.-W. Kim).

0020-0255/$ - see front matter
doi:10.1016/j.ins.2007.08.022

Table 1
An example of network traffic data gathered by Ethereal

Timestamp	Source address	Source port	Destination address	Destination port
13:37:11.950966	180.1.1.1	36872	amazon.com	www
13:37:11.954474	amazon.com	www	180.1.1.1	36872
13:37:22.384472	180.1.1.1	36915	192.168.1.3	telnet
13:37:22.385327	192.168.1.3	telnet	180.1.1.1	36915

Table 2
An example of sequential patterns that can be discovered from network traffic data

Sequential pattern 1	Receiving data from 192.168.1.254 → Sending data to 192.168.1.254 → Sending data to 192.168.1.254 → Sending data to 192.168.1.254
Sequential pattern 2	Receiving data from amazon.com → Sending data to amazon.com

Table 1 shows an example of network traffic data gathered by Ethereal.[1] A row in the table represents an individual network traffic datum and consists of its source address, source port, destination address, destination port, and timestamp.

Network traffic data have the following characteristics in comparison with other data: First, various kinds of data exist since all the computers connected to the Internet can potentially produce network traffic data. Second, a huge amount of network traffic data accumulates due the frequent exchanges among many computers sending/receiving information. Third, the network traffic data to be analyzed are generally scattered over a large number of sites.

Various data mining techniques such as association rules [19] and clustering [6] can be applied for analyzing network traffic data. Sequential pattern mining [12,15,27], however, is the most useful since the order of events has an important meaning in network traffic data. Table 2 shows an example of supposed sequential patterns that can be discovered from network traffic data. Here, sequential pattern 2 says that numerous sites send data to "amazon.com" right after receiving data from "amazon.com".

Network traffic data contain detailed information of Internet usage for every user, which informs that a user accesses a site at a time specifically. Herein, data mining on network traffic data inherently has the problem of compromising privacy of network users. Therefore, it requires sophisticated techniques for hiding or reforming users' private information during a data gathering process. Moreover, these techniques should not sacrifice the correctness of mining results.

Privacy-preserving data mining is a new research area that aims to mine data while guaranteeing the privacy of individual users [2,3,5,7,8,13,16,18,24,25,30,32]. Recently, there have been many research efforts performed in this area. Most methods proposed in prior studies, however, manage data in a few sites or deal with a small number of distinct types of data. Thus, these methods are not appropriate for mining network traffic data since they suffer from inaccuracy and low performance.

This paper proposes an efficient mining method of sequential pattern mining that preserves privacy while solving the problems of inaccuracy and low performance encountered in prior methods. In order to discover frequent items (i.e., a pattern of length 1) without compromising privacy, the proposed method uses the N-repository server model that operates as a single mining server. Also, it maintains *meta tables* in each site so as to quickly determine whether candidate frequent patterns have ever occurred in the site, thereby making the whole mining process highly efficient.

The procedure for finding frequent items with the N-repository server model is as follows: first, every site partitions its own network traffic data into N groups by using a hash function and encrypts each group by a unique encryption key assigned to the group. Then, it sends each encrypted group to one of N servers. Note

[1] http://www.ethereal.com/.

that this server cannot decrypt the group since it does not have the corresponding decryption key. A server determines frequent items by totalizing the occurrences of each item received. For decrypting each frequent item thus discovered, the server sends the item to another server that has its corresponding decryption key. The N servers all perform the decryption process for items received and then make one coordinating server totalize the occurrences of each candidate frequent item in order to find the real frequent items.

The coordinating server generates candidate patterns by combining those frequent items found and queries every site to assess whether each candidate pattern occurs in the site. In order to quickly recognize the occurrences of candidate patterns, each site maintains meta tables that store two things: (1) pairs of frequent items that occur together within a pre-specified interval and (2) frequent patterns that occurred in the site. In the meta tables, 1 and 0 indicate whether the frequent pattern has occurred or not, respectively. Also, each site sends the coordinating server these values perturbed by the probability p. For each candidate pattern, the server totalizes the occurrences of 1 and 0 and then determines the candidate frequent pattern by computing the real distribution of 1 and 0 using the frequencies and probability p.

In this paper, we discuss solutions to the problems that occur in previous methods. We propose a novel method for sequential pattern mining on network traffic data. The proposed method preserves site privacy and guarantees the correctness of the mining results. The method discovers frequently occurring network traffic patterns (i.e., frequent items) while hiding site information in two ways: (1) it employs the N-repository server model, which forces multiple servers to behave as a single mining server; (2) it uses the *retention replacement* technique, which changes the answer according to a given probability. Also, the method maintains *meta tables* in each site so as to quickly determine whether candidate patterns ever occurred in the site, thereby making the overall mining process highly efficient.

The paper is organized as follows: Section 2 reviews previous work related to privacy preserving data mining. Section 3 defines the problem this paper is trying to solve in more detail. Section 4 presents our detailed methodology. Section 5 verifies the correctness and the efficiency of the proposed method via a variety of experiments performed on real-world network traffic data. Finally, Section 6 summarizes the paper and suggests possible applications for the proposed method.

2. Related work

Sequential pattern mining discovers frequently occurring patterns from large sequence databases [1]. Srikant and Agrawal gave a formal description of the problem of mining generalized sequential patterns [28]: (1) time constraints are added to specify a minimum and/or maximum time period between adjacent items; (2) a rigid definition of a transaction is relaxed to allow a pattern to stretch over two adjacent transactions; (3) sequential patterns are allowed to include items across all levels of a user-specified taxonomy. Also, they proposed the GSP (general sequential pattern) algorithm for mining such generalized sequential patterns.

Following Srikant's work, many studies have been performed to determine a more efficient method of discovering sequential patterns [4,10,14,17,20,23]. A typical method is the pattern-growth approach that is based on the divide-and-conquer concept [10,23]. It recursively projects a sequence database into a set of smaller projected sequence databases, grows sequential patterns in each projected database by exploring only locally frequent patterns, and then produces the final results by combining local frequent patterns. Kum et al. proposed a method called *ApproxMap* for approximate sequential pattern mining [14]. It uses clustering as a preprocessing step in order to group similar sequences, and then mines the underlying consensus patterns in each cluster through multiple alignments.

The problem of pushing various constraints deep into sequential pattern mining was addressed in [9,22]. Also, the problem of incremental updates in sequential pattern mining was dealt with in other studies [21,31]. To reduce many of the meaningless results that are produced, the concept of *frequent closed patterns*, those containing no super-patterns with the same support, was introduced in [29] along with efficient discovery methods.

Lee et al. proposed a method that applies sequential pattern mining to intrusion detection [15]. The method first extracts *frequent episode rules* by mining network traffic data, builds an intrusion detection model with those rules, and detects network intrusions based on this model. In this model, a site corresponds to a sequence to be mined and network traffic data occurring in the site correspond to an item belonging to a transaction.

Recently, there have been several research efforts to find effective solutions to intrusion detection by mining network traffic data [6,12,19,27].

Network traffic data contain detailed information about the Internet usage of individual users. Thus, data mining on network traffic data may inherently compromise the privacy of network users. Therefore, it requires sophisticated techniques for masking or reforming users' private information.

Clifton and Marks [5] first raised the issue of privacy in data mining and inspired subsequent studies [2,3,7,8,13,16,18,24,25,30,32] that aimed to solve the problem. We can classify these studies into two categories.

Approaches in the first category avoid violating privacy by perturbing the values of individual items [2,3,7,18,25]. For accurate data mining, they maintain the value distribution of perturbed items kept identical to the original one. These approaches work well with those data mining algorithms that use the probability distribution rather than individual items.

In the method proposed in [2], in order to preserve privacy, each site changes the original value of an individual item before sending the value to the server by adding an arbitrary value selected from a given probability distribution. The server builds a decision tree by referencing the actual value distribution, which is reconstructed by using the probability distribution.

Another method called *retention replacement* [3,25] perturbs and reconstructs data in gathering and mining, respectively, for privacy preservation. For every datum whose element represents 0 or 1, each site sends the original element value with probability p and the perturbed one with probability (1-p). For gathered data, the server counts the total numbers of 1's and 0's and then estimates the original numbers of 1's and 0's from reconstructed data, where an element is retained with probability p and replaced with probability (1-p). This method is limited, because it is applicable only to boolean data.

Some later studies [3,7] tried to apply those two methods to applications with various types of data, but these produced lower accuracy in mined results as the number of possible values increases.

The method proposed by Rizvi and Haritsa [25] uses 1 and 0 to represent whether the data occur or not, respectively. Next, it gathers all data perturbed by the *retention replacement* and determines the support by computing 1's distribution. Then, it finds frequent items by using the support and discovers association rules by extending them. This method can be applied in cases where pre-determined item types occur. Considering network traffic data where a large number of item types occur, we can hardly determine all the item types in advance. Also, this method finds frequent patterns via a whole database scan and thus is very inefficient since network traffic data are huge. Therefore, we cannot directly apply this method to the discovery of sequential patterns from network traffic data with privacy preservation.

Liu et al. [18] applied a data perturbation method to data warehouses for range queries without allowing access to individual data values. This method supports only 'summation' queries and therefore has limited applicability.

In the methods belonging to the second category [8,13,16,24,30,32], a site participates in the data mining process and directly handles data that can be compromised while a server produces the final result by totalizing the intermediate results obtained from all the sites. A typical example is the method proposed by Kantarcioglu and Clifton [13]. This method securely collects local frequent itemsets from sites by using commutative encryption, obtains global frequencies of all the data by using a secure sum which uses a random number, and finally discovers association rules. For performing commutative encryption and a secure sum, this method has to send data serially in the cycle of sites. This method is time intensive in cases with a large number of sites. Fukasawa et al. [8] improved the efficiency and security of this method. However, the improved method still has cycling communications.

Zhong [32] proposed two privacy preservation methods that can find frequent itemsets from either vertically or horizontally distributed databases. These methods are designed for the semi-honest model among two or more parties and employ a probabilistic public-key encryption for secure protocols. However, these methods cannot be applied to databases that are both vertically and horizontally distributed. Furthermore, since they examine every candidate itemset individually, they are not practical in large databases.

Zhan et al. proposed a method for sequential pattern mining with privacy preservation [30]. This method mainly targets a distributed database environment where vertical partitioning without duplication is employed. In our situation, duplicated data could occur in more than one site since multiple PCs can access the same Internet site. Therefore, this method is inapplicable to network traffic data in the current form.

In the method proposed in [16], a secure protocol is used for mining a decision tree classifier from distributed sites. Pinkas [24] showed how protocols for secure distributed computation can be employed for privacy preservation; however, he also pointed out that the performance of the proposed protocols should be improved.

In summary, the methods proposed in prior studies have problems in applications with large amounts of network traffic data. First, due to a variety of data types, previous methods are not directly applicable and cannot produce accurate mining results. Second, since a large number of sites exist and data can be duplicated, previous methods targeted for a distributed database environment have practical limitations.

3. Problem definition

Network traffic data are normally gathered by a tcp/ip data capture program such as Ethereal. As shown in Table 1, the information obtained from Ethereal consists of its source address, source port, destination address, destination port, and timestamp. In this paper, we aim at finding sequential patterns, as shown in Table 2, from network traffic data without disclosing data in each site. First, we simplify the network traffic data in Table 1 to match those in Table 3. The example in Table 3 represents the network traffic data sent/received by site "180.1.1.1", where "out" denotes the sending site and "in" does the receiving site. We note that Table 1 includes port information in the network traffic data that Table 3 does not include. This reduces the types of possible network traffic data by simplifying them, thereby making the probability of appearances of frequent patterns increase considerably.

In order to find a temporal relationship among the events in network traffic data thus reconstructed, we can apply sequential pattern mining methods [12,15,27] after representing each traffic data tuple as an item. At this point, the maximum time interval is set for deciding whether two adjacent items have meaningful temporal relationship: (1) without such a setting, the number of sequential patterns to be considered becomes too high; (2) it is hard to say that two adjacent items whose time interval is large are mutually related. Herein, we impose a restriction that two adjacent items should have a time interval smaller than or equal to a predefined *MaxGap* value to be regarded as related.

We formulate the problem to be solved as follows: given t sites $T_1, T_2, \ldots, T_t$, the maximum time interval *MaxGap*, and the minimum support *MinSup*, we discover all the sequential patterns, each of which has a support larger than *MinSup* and a time interval between any pair of adjacent items equal to or smaller than *MaxGap*. It is assumed that a site stores network traffic data in the form of Table 3 but does not throw them open to the public.

A mining process should also satisfy the conditions for preserving privacy in every site. Let us denote a set of sites where network traffic has occurred as E and a set of network traffic data as I. In a mining process, an element e_j in E is opened since it participates in the mining process; also, an element i_k in I is opened since it is a communication record and should be contained in a sequential pattern resulting from the mining process. However, a pair of (e_j, i_k), which denotes that a site e_j has been connected to an IP address i_k, should not be opened in a mining process, as a condition for privacy preservation.

4. Proposed method

In this section, we propose an efficient and practical method for solving the problems discussed in the previous section. We first outline the overall mining process in Section 4.1 and explain the process to discover

Table 3
An example of network traffic data reconstructed

Timestamp	In/out	Address
13:37:11.950966	Out	amazon.com
13:37:11.954474	In	amazon.com
13:37:22.384472	Out	192.168.1.3
13:37:22.385327	In	192.168.1.3

frequent patterns of length 1 (i.e., frequent items or large 1-sequences) in Section 4.2. We then explain the process to find out frequent patterns longer than 1 in Section 4.3. Finally, in Section 4.4, we describe the structure and usage of the *meta tables* maintained in each site to quickly determine whether candidate patterns have ever occurred or not.

4.1. Overall mining process

The proposed mining process consists of four phases as shown in Fig. 1. The first phase utilizes the N-repository server model to safely discover F_1, (or frequent items), large 1-sequences. The second phase generates C_{k+1}, a set of all candidate patterns of length $k+1$, by self-joining large k-sequences (i.e., F_k). k is initialized to 1 when the second phase is executed for the first time. If C_{k+1} is empty, we enter into the final phase. Otherwise, we enter into the third phase. For each candidate pattern in C_{k+1}, the third phase sends every site the query asking whether the candidate pattern has ever occurred in the site or not. After receiving answers from all sites, the third phase judges whether each candidate pattern is frequent or not, and then constructs F_{k+1}, large $k+1$ sequences, with the candidate patterns judged as frequent. The third phase then increases k by 1 and calls the second phase. The final phase prints all frequent patterns, $F_1, F_2, \ldots, F_k$, and stops the mining process (see Fig. 2).

4.2. Finding frequent items using N-repository server model

The proposed N-repository server model finds frequent items, F_1, without compromising the privacy constraint by concealing the linkage between the site identifier and the traffic data, (e_j, i_k), during the mining process. More specifically, it obscures their linkage by encrypting the traffic data, i_k, at the first step and by aggregating the site identifiers, e_j, at the second step.

The proposed N-repository server model consists of N servers, $\{S_1, S_2, \ldots, S_N\}$, and N pairs of encryption keys and decryption keys, $\{(EK_1, DK_1), (EK_2, DK_2), \ldots, (EK_N, DK_N)\}$. Each site has all encryption keys but

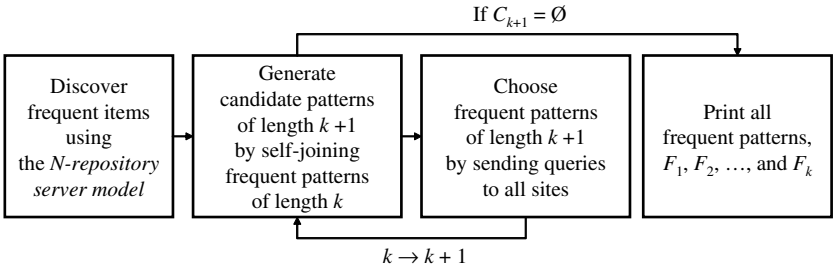


Fig. 1. Overall mining process.

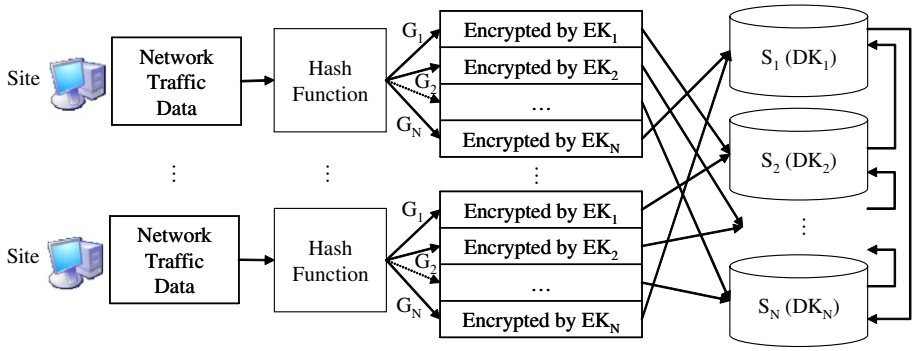


Fig. 2. Process to distribute and encrypt the traffic data using a hash function and encryption keys, respectively.

only the server S_i has the decryption key DK_i ($1 \leqslant i \leqslant N$). To find frequent items safely, the N-repository server model operates as follows:

1. Each site classifies the items (i.e., the traffic data) organized as in Table 3 into N groups, $\{G_1, G_2, \ldots, G_N\}$, using a hash function.
2. Each site encrypts the items in G_i with the encryption key EK_i ($1 \leqslant i \leqslant N$).
3. Each site sends the encrypted items in G_i to server S_{i+1} ($1 \leqslant i \leqslant N-1$) and the encrypted items in G_N to server S_1. The privacy of each site is preserved since no servers have the keys to decrypt the items they receive.
4. Each server performs the aggregation on the encrypted items to obtain the number of occurrences of each encrypted item and then picks up the encrypted *frequent* items, which are the encrypted items whose numbers of occurrences are larger than a system-defined threshold.
5. Each server S_i sends encrypted frequent items to server S_{i-1} ($2 \leqslant i \leqslant N$) and server S_1 sends encrypted frequent items to server S_N.
6. Each server S_i decrypts the received items with its decryption key DK_i and then reports the frequent items to a principal mining server. The privacy of each site is still preserved even after decryption because the linkages between sites and frequent items have been already obscured by aggregation.

In our N-repository server model, an identical hash function is used in every site, and, by the property of the hash function, the items with the same content congregate in the same group and every item has a corresponding group. Therefore, it is obvious that the union of frequent items discovered within each group is identical to the set of frequent items discovered from the entire data set without grouping.

We assume that the servers in our model operate in a *semi-trusted* operation model. In the semi-trusted operation model, servers may try to acquire private data but do not cooperate with other servers when doing so. This semi-trusted operation model is common in real environments where one wants to obtain the result of a computation but is not willing to offer one's own data to others [30].

4.3. Finding frequent patterns longer than one

After finding all frequent items (i.e., large 1-sequences) using the N-repository server model, we have to discover sequentially frequent patterns longer than one. At first, one of N servers is elected as a principal mining server to which all the other servers send the frequent items they discovered. Let F_1 be the set of all frequent items collected at the principal mining server. To discover all frequent patterns longer than one, the principal mining server assigns 1 to variable k and executes the following steps.

1. It produces C_{k+1}, the set of candidate patterns of length $k+1$, by self-joining F_k in the same way as the *Apriori* algorithm [1]. It executes step 5 if C_{k+1} is empty. Otherwise, it executes step 2.
2. For each candidate pattern CP in C_{k+1}, it sends every site T a query asking whether CP has ever occurred in T or not.
3. Each site T sequentially inspects its own traffic data or the meta tables, which will be described in Section 4.4, to determine CP's occurrence or non-occurrence in T. An actual answer to the query would be 1 if CP has ever occurred in T and 0 if CP has never occurred in T. However, to preserve the privacy of the site, the actual answer is perturbed by the application of *retention replacement* [25,3]. More precisely, the query is answered with an actual answer by a given probability p and with the reverse of an actual answer by the probability 1-p.
4. For each query, the principal mining server aggregates the count of the sites that answered 1 and the count of the sites that answered 0. Then, using the two counts and the probability p, it conjectures the number of sites whose actual answers were 1 and the number of sites whose actual answers were 0. It then compares the number of sites whose actual answers are supposed to be 1 with *MinSup*, a given minimum support count. It then constructs F_{k+1}, the set of frequent patterns of length $k+1$, by choosing from C_{k+1} only the candidate patterns whose estimated numbers of occurrences are at least *MinSup*. It finally increases k by 1 and calls step 1.

5. When it reaches this step, C_{k+1} is empty and thus no more candidate patterns can be generated. Therefore, it prints all the frequent patterns it has discovered so far (i.e., $F_1, F_2, \ldots, F_k$) and then stops the execution of the algorithm.

4.4. Meta tables to quickly determine the occurrence or non-occurrence of candidate patterns

In this Section, we describe the structures of the *meta tables* maintained in each site to quickly determine whether or not candidate patterns have ever occurred in the site. Before jumping into the details, let us explain why extra *meta tables* are needed in our mining process. In the original *Apriori* algorithm, patterns of length k can be regarded as candidate patterns only when all of their sub-patterns of length $k-1$ are frequent. For example, pattern $\langle A, B, C\rangle$ can be treated as a candidate pattern only when all of its sub-patterns of length 2, $\langle A, B\rangle$, $\langle B, C\rangle$, and $\langle A, C\rangle$, are frequent. However, in sequential pattern mining with time constraints, even the patterns containing infrequent sub-patterns can be treated as candidate patterns if all of their sub-patterns occurring *contiguously* in the underlying patterns are frequent. For example, pattern $\langle A, B, C\rangle$ can be treated as a candidate pattern in the sequential pattern mining if both $\langle A, B\rangle$ and $\langle B, C\rangle$ are frequent, regardless whether $\langle A, C\rangle$ is frequent or not. Therefore, compared with the mining techniques based on the original *Apriori* algorithm, the sequential pattern mining algorithms with time constraints impose fewer requirements for patterns to be treated as candidate patterns. As a result, more candidate patterns are generated in the sequential pattern mining with time constraints and, to accelerate the overall mining process, it is crucial to handle each candidate pattern efficiently.

When the principal mining server sends each site T a query asking if a candidate pattern CP has ever occurred in T, site T sequentially inspects its own traffic data until it detects the occurrence of CP. In the best case, site T can detect the occurrence of CP after examining the first few items of its traffic data. However, in the worst case, site T can make such decision after inspecting all the items of its traffic data. In this paper, we propose employment of special-purpose *meta tables* in each site T to speed up the process of determining the occurrence or non-occurrence of CP in T. More specifically, we extract from T's traffic data all pairs of items whose time gaps are not larger than *MaxGap*, a system-defined maximum time gap, and then store them into the *meta tables* of T. In addition, we store into the *meta tables* the candidate patterns whose occurrences are inquired by the principal mining server and then whose occurrences are detected in site T. Let us first describe the *meta tables* for storing all pairs of items whose time gaps do not exceed *MaxGap*.

4.4.1. Meta tables for storing pairs of items satisfying MaxGap

Let m denote the number of frequent items discovered by using the N-repository server model. At first, the principal mining server sends out the list of all frequent items to each site T. Then, site T lexicographically sorts the frequent items it received and assigns each frequent item the corresponding lexicographic order. Note that all the lexicographic orders must be within the range from 1 to m. Site T then stores the name and lexicographic order of each frequent item into the meta table called FreqItems. Table FreqItems consists of two columns, ItemName and Order. Given a frequent item, columns ItemName and Order store its name and lexicographic order, respectively.

The second meta table maintained in each site is OccTs_OccBits. This table consists of three columns, Order, OccTs, and OccBits. For each frequent item FI found in the traffic data of T, column Order stores the lexicographic order of FI, and column OccTs stores the timestamp at which FI occurred, and column OccBits stores a bit-vector of length m whose ith bit indicates whether or not the frequent item of the lexicographic order i has ever occurred within *MaxGap* after the occurrence of FI. We denote the ith bit of column OccBits as OccBits(i). Table OccTs_OccBits can be constructed by scanning the entire traffic data stored in site T. When a frequent item is found during the scan, a new tuple is created and then inserted into table OccTs_OccBits. Therefore, the number of tuples in table OccTs_OccBits is same as the number of occurrences of frequent items in T.

The third meta table maintained in each site is OccCnts. Table OccCnts consists of $m + 1$ columns, Order, $\mathsf{Cnt}_1, \mathsf{Cnt}_2, \ldots, \mathsf{Cnt}_m$. Table OccCnts has a tuple for each frequent item and therefore contains m tuples. Let us consider the ith tuple of table OccCnts. It has i as a value of column Order. As a value of column Cnt_j, it has

the number of occurrences of the frequent item of order j whose timestamps are within *MaxGap* after the occurrences of the frequent item of order i. The ith tuple of table OccCnts can be populated by executing the following SQL statement:

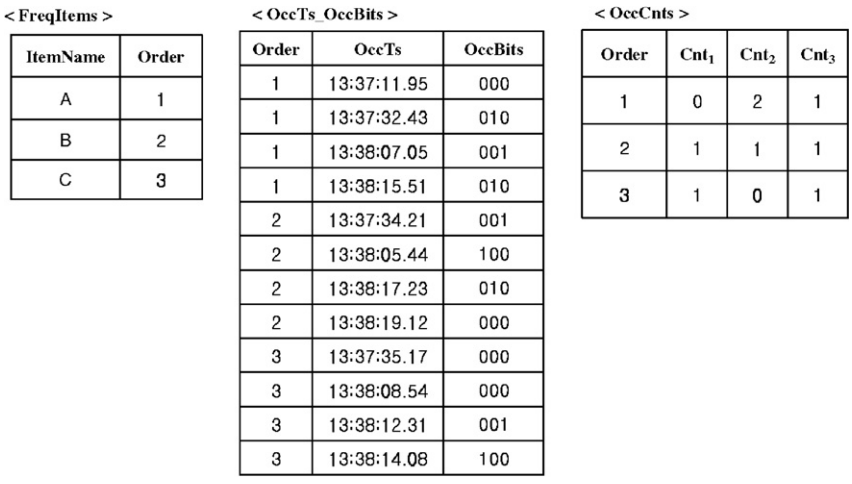
```
insert into OccCnts (Order, Cnt_1, Cnt_2, ..., Cnt_m) values (
    i,
    (select count(*) from OccTs_OccBits
    where Order = i and OccBits(1) = 1),
    (select count(*) from OccTs_OccBits
    where Order = i and OccBits(2) = 1),
    ...
    (select count(*) from OccTs_OccBits
    where Order = i and OccBits(m) = 1)
);
```

An example of *meta tables* maintained within a single site is shown in Fig. 3. Table FreqItems has three frequent items, *A*, *B*, and *C*, and their orders are 1, 2, and 3, respectively. Table OccTs_OccBits has 12 tuples and its key consists of two columns, column Order and column OccTs. To understand the functions of its columns, let us consider its second tuple. The second tuple expresses that the frequent item of order 1, which is *A*, occurred at timestamp 13:37:32.43 and only the frequent item of order 2, which is *B*, occurred within *MaxGap* from 13:37:32.43. Table OccCnts has three tuples each of which is distinguished by column Order. To understand the functions of its columns, let us consider its first tuple. It indicates that, within *MaxGap* after the occurrence of the frequent item of order 1 (i.e., item *A*), the frequent item of order 1 (i.e., item *A*) occurred 0 times, the frequent item of order 2 (i.e., item *B*) occurred twice, and the frequent item of order 3 (i.e., item *C*) occurred once.

If we use these three *meta tables*, we can quickly determine whether or not candidate patterns have ever occurred in sites without sequentially scanning the traffic data. In Section 5, we compare the performance of the method utilizing these three *meta tables* with the performance of the sequential scanning method.

4.4.2. Determining the occurrences or non-occurrences of candidate patterns

The algorithm to determine the occurrences or non-occurrences of candidate patterns using the *meta tables* has 4 steps as shown in Fig. 4. A candidate pattern is divided into several sub-patterns in step 1 and the execution orders of the sub-patterns are determined in step 2. In step 3, the sub-patterns are executed one by one according to their execution orders and their results are joined with the previous results. Finally, in step 4, the occurrence or non-occurrence of the candidate pattern is determined by inspecting the final join result.



< FreqItems >

ItemName	Order
A	1
B	2
C	3

< OccTs_OccBits >

Order	OccTs	OccBits
1	13:37:11.95	000
1	13:37:32.43	010
1	13:38:07.05	001
1	13:38:15.51	010
2	13:37:34.21	001
2	13:38:05.44	100
2	13:38:17.23	010
2	13:38:19.12	000
3	13:37:35.17	000
3	13:38:08.54	000
3	13:38:12.31	001
3	13:38:14.08	100

< OccCnts >

Order	Cnt_1	Cnt_2	Cnt_3
1	0	2	1
2	1	1	1
3	1	0	1

Fig. 3. An example of *meta tables* maintained within a single site.

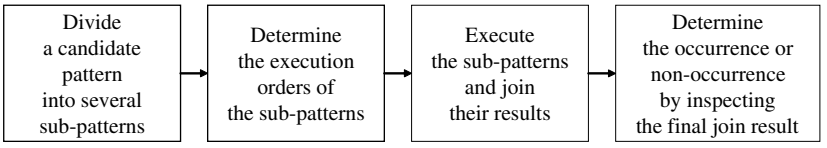


Fig. 4. Process to determine the occurrences or non-occurrences of candidate patterns using *meta tables*.

1. Divide a candidate patternLet $CP_n = \langle I_1, I_2, \ldots, I_n \rangle$ denote a candidate pattern with n items. We first divide CP_n into $n-1$ sub-patterns each of which consists of two adjacent items of CP_n. The jth sub-pattern of CP_n is denoted as $SP_j = \langle I_j, I_{j+1} \rangle$ $(j = 1, 2, \ldots, n-1)$.
2. Determine the execution orders of the sub-patternsSub-patterns are executed on the *meta tables*, and their results are joined with those of other sub-patterns. The sizes of their intermediate results depend on the execution orders of the sub-patterns. If we are able to discover the optimal execution orders that minimize the sizes of the intermediate results, we can determine the occurrence or non-occurrence of a candidate pattern earlier. The simplest way to determine the optimal execution orders is to consider all possible combinations of execution orders and to choose the one that will produce the smallest intermediate results. However, there are $(n-1)!$ distinct combinations of execution orders for $n-1$ sub-patterns and thus such a simple approach is not scalable to large n. Therefore, we employ the following *greedy* algorithm which quickly discovers near-optimal execution orders of sub-patterns.
   (a) We choose the sub-pattern that will have the smallest result set size, and we let 1 be its execution order. We then assign 1 to variable k.
   (b) Let us assume that the execution order of sub-pattern SP_j has just been decided as k. To decide the sub-pattern of execution order $k+1$, we decrease j' from $j-1$ to 1 one by one until we find the sub-pattern $SP_{j'}$ whose execution order is not yet decided. Also, we increase j'' from $j+1$ to $n-1$ one by one until we find the sub-pattern $SP_{j''}$ whose execution is not yet decided.
   (c) If neither $SP_{j'}$ nor $SP_{j''}$ exists, we conclude that the execution orders of all sub-patterns have already been decided. Therefore, we stop the execution of the *greedy* algorithm. However, if $SP_{j''}$ does not exist but $SP_{j'}$ does exist, then we let $k+1$ be the execution order of $SP_{j'}$. On the contrary, if $SP_{j'}$ does not exist but $SP_{j''}$ does exist, then we let $k+1$ be the execution order of $SP_{j''}$. If both $SP_{j'}$ and $SP_{j''}$ exist, then we choose the one that will have a smaller result set size and let $k+1$ be its execution order. If their result set sizes will be same, then we choose the one farther from the corresponding end. That is, if $(j'-1) \geqslant (n-1-j'')$, then we choose $SP_{j'}$. Otherwise, we choose $SP_{j''}$. This reduces the possibility of the absence of either $SP_{j'}$ or $SP_{j''}$ in the next step and therefore enables us to obtain a better combination of execution orders.
   (d) We increase k by one and return to step 2(b).

In the middle of this *greedy* algorithm, there is a step to calculate the result set sizes of sub-patterns. The result set size of sub-pattern $SP_j = \langle I_j, I_{j+1} \rangle$ is equal to the number of occurrences of item I_{j+1} within *MaxGap* after the occurrences of item I_j. The result set size of a sub-pattern can be easily obtained by using two *meta tables*, FreqItems and OccCnts, whose structures were explained in Section 4.4.1. More specifically, we first obtain the lexicographic orders of item I_j and I_{j+1} by using table FreqItems. Let p and q denote their lexicographic orders, respectively. We then execute the following SQL statement on table OccCnts to directly obtain the result set size of sub-pattern SP_j.

select Cnt_q from OccCnts where Order $= p$.

3. Execute the sub-patterns and join their results.
   According to the execution orders obtained in step 2, we execute all sub-patterns one by one while joining their intermediate results. That is, for each k from 1 to $n-1$, we execute the following steps.
   (a) For the two items of the sub-pattern whose execution order is k, we find their lexicographic orders using a meta table FreqItems. Let p and q be the lexicographic orders of the preceding item and the succeeding item, respectively.

(b) We execute the following SQL statement on table OccTs_OccBits to obtain the result set RS_k of the sub-pattern of execution order k.

```
select p, OccTs, q   // p and q are not column names but constants
into RS_k
from OccTs_OccBitswhere Order = p and OccBits(q) = 1;
```

(c) We join the result set RS_k with the table JRS_{k-1}, the intermediate result set obtained by sequentially joining all the result sets of sub-patterns of execution orders from 1 to $k-1$, producing a new intermediate result set JRS_k. For a simpler explanation, let us rename the tables to be joined as follows. If the sub-pattern of execution order k is on the left of the sub-patterns of execution orders from 1 to $k-1$, then we rename the sub-pattern of execution order k as TA and the intermediate result set JRS_{k-1} as TB. Otherwise, we rename the sub-pattern of execution order k as TB and the intermediate result set JRS_{k-1} as TA. Then, the conditions for a tuple *ta* of table TA to be joined with a tuple *tb* of table TB are like the following:

- *Join condition* 1: The last item of tuple *ta* must be identical to the first item of tuple *tb*.
- *Join condition* 2: The gap from the timestamp of *tb*'s first item to the timestamp of *ta*'s last item must not be larger than *MaxGap*.

(d) We check if the join result JRS_k is empty. If so, we proceed to step 4. Otherwise, we increase k by one and return to step 3(a).

4. Determine the occurrence or non-occurrence of a candidate pattern
   We check if the final result set of step 3 is empty. If so, we conclude that the candidate pattern in consideration has never occurred in this site. Otherwise, we conclude that there is at least one occurrence of the candidate pattern in this site.

To illustrate the above algorithm, let us consider the *meta tables* shown in Fig. 3 and a candidate pattern $\langle A, B, C\rangle$, with the assumption that *MaxGap* is 10. In the first step, the candidate pattern is divided into two sub-patterns, $\langle A, B\rangle$, and $\langle B, C\rangle$. In the second step, the execution orders of two sub-patterns are decided by calculating their result set sizes using table OccCnts. Since the result set sizes of sub-patterns $\langle A, B\rangle$ and $\langle B, C\rangle$ are 2 and 1, respectively, their execution orders are determined as 2 and 1, respectively. In the third step, two sub-patterns are executed according to their execution orders. The sub-pattern of execution order 1, which is $\langle B, C\rangle$, produces $RS_1 = (\langle 2, 13{:}37{:}34.21, 3\rangle)$, and the sub-pattern of execution order 2, which is $\langle A, B\rangle$, produces $RS_2 = (\langle 1, 13{:}37{:}32.43, 2\rangle, \langle 1, 13{:}38{:}15.21, 2\rangle)$. Right after obtaining RS_2, we join it with JRS_1, which is the same as RS_1 in this example, and obtain the final result $JRS_2 = (\langle 1, 13{:}37{:}32.43, 2, 13{:}37{:}34.21, 3\rangle)$. Since the final result set is not empty, we decide that the candidate pattern $\langle A, B, C\rangle$ occurred in this site at least once.

4.4.3. Meta tables to quickly judge the non-occurrence of a candidate pattern

The *Apriori* algorithm joins frequent patterns of length n with themselves to generate candidate patterns of length $n+1$. That is, it joins two frequent patterns of length n, $\langle I_1, I_2, \ldots, I_n\rangle$ and $\langle I_2, I_3, \ldots, I_{n+1}\rangle$, to produce a candidate pattern of length $n+1$, $CP_{n+1} = \langle I_1, I_2, \ldots, I_{n+1}\rangle$. The lengths of candidate patterns increase successively by this self-joining process.

Let $\{CP_n\}$ denote the set of candidate patterns of length n delivered to the site. Also, let $\{CP'_n\}$ denote the set of candidate patterns in $\{CP_n\}$ whose occurrences were detected in the site. Now, let us consider a sub-pattern of length $n+1$ (i.e., CP_{n+1}) most recently delivered to the site. If we break CP_{n+1} into two sub-patterns of length n, $CP_{n+1}[1 \ldots n]$ and $CP_{n+1}[2 \ldots n+1]$, then both of them are certainly elements of $\{CP_n\}$. The prerequisites for CP_{n+1} to occur in the site are the occurrences of both $CP_{n+1}[1 \ldots n]$ and $CP_{n+1}[2 \ldots n+1]$. Therefore, if either $CP_{n+1}[1 \ldots n] \notin \{CP'_n\}$ or $CP_{n+1}[2 \ldots n+1] \notin \{CP'_n\}$ are satisfied, then we can quickly recognize the non-occurrence of CP_{n+1} without looking up the *meta tables* described in Section 4.4.1.

We implement this idea by maintaining a meta table named OccCandPatt in each site. Meta table OccCandPatt consists of two columns, Len and Patt. For each candidate pattern whose occurrence was detected in the site, column Len stores its length and column Patt stores its string representation. When the site receives a candidate pattern of length $n+1$ (i.e., CP_{n+1}), it first consults meta table OccCandPatt to detect its non-occurrence as early as possible. The detailed procedure is as follows.

1. We break CP_{n+1} into two sub-patterns of length n, $CP_{n+1}[1 \ldots n]$ and $CP_{n+1}[2 \ldots n+1]$.
2. We execute the two following SQL statements, producing two temporary tables TA and TB:
   select * into TA from OccCandPatt
   where Len $= n$ and Patt $= CP_{n+1}[1 \ldots n]$;
   select * into TB from OccCandPatt
   where Len $= n$ and Patt $= CP_{n+1}[2 \ldots n+1]$;
3. If either TA or TB is empty, we declare the non-occurrence of the candidate pattern CP_{n+1}. Otherwise, we entrust the decision of its occurrence or non-occurrence to the algorithm described in Section 4.4.2. If its occurrence is detected afterwards by the algorithm of Section 4.4.2, then its length and its string representation are stored into table OccCandPatt.

The above algorithm enables the swift assessment of the non-occurrence of a candidate pattern but continually increases the size of table OccCandPatt. However, note that the above algorithm requires only the candidate patterns of length n in order to determine the non-occurrence of a candidate pattern of length $n+1$. Therefore, when the site receives from the principal mining server a candidate pattern of length $n+1$ for the first time, it removes the candidate patterns of length $n-1$ from table OccCandPatt. As a result, we do not have to worry about the continual growth of the size of table OccCandPatt.

4.5. Discussions

4.5.1. Practicality

Over the Internet, a large number of sites exist and data may be duplicated. Applying the previous methods directly to this environment is impractical. The performance of the method using commutative encryption and secure sum [13] seriously deteriorates with a large number of sites. Also, with a method targeted for a vertically distributed database environment [30], parallel processing is inapplicable since it has to perform sequential cycling operations over distributed databases.

Rather than using a protocol with sequential cycling operations, our method forces a number of sites to process queries sent by a mining server in a parallel fashion without any dependencies. Its total elapsed time depends only on the site that requires the largest processing time. Therefore, our method can be applied to situations with a large number of sites.

4.5.2. Accuracy

Network traffic data have a variety of data types. In this case, the previous methods proposed in [2,3,7,25] suffer from inaccurate mining results because they perform data mining by using the retention replacement or by perturbing the values in individual items. These methods can be applied only to the situations where item types are known in advance. However, we can hardly know all the item types beforehand because there are a large number of item types in network traffic data. Thus, the accuracy of the prior methods decreases with the number of item types.

Our method does not suffer from this problem because it employs the N-repository server model, which does not use retention replacement or perturbation in finding frequent items. Therefore, it is useful in cases where item types cannot be pre-determined in advance as in network traffic data.

4.5.3. Performance

The Apriori algorithm [1] has to scan an entire database from a disk in each step to discover frequent patterns. This requires a large processing overhead. Thus, considering the large amount of network traffic data to be mined, we need an efficient mechanism to determine whether candidate items have previously occurred or not.

To that end, our method employs meta tables that store all the pairs of frequent items, thereby quickly determining whether candidate items have occurred without incurring disk accesses to original network traffic data. Owing to the meta tables, the performance of the proposed method improves considerably.

In Section 5, we verify the superiority of the proposed method in all the aspects above via extensive experiments.

5. Performance evaluation

This section shows the superiority of the proposed method via performance evaluation with extensive experiments. Sections 5.1 and 5.2 describe the environment and parameter settings for experiments, respectively. Sections 5.3–5.5 present and analyze the results.

5.1. Environment for experiments

In our experiments, we collected 5,024,295 network traffic data by using Ethereal, a program to capture packets, during 5 days in October 2005. From them, we extracted 747,000 network traffic data related to 736 IP addresses, including the occurrence time, the status of sending or receiving, and the target address, which are necessary for experiments involving sequential pattern mining. We used them as query and data sets. The average inter-arrival time was 462.38 ms.

We compared the performances of three methods: Naive, OccTs, and OccTs+OccCandPatt. In order to discover the frequent itemset F_1, Naive uses the *retention replacement* for all traffic data. Furthermore, it scans the original traffic data sequentially to verify whether every candidate is actually frequent. OccTs discovers F_1 by using the N-repository server model. OccTs decomposes a candidate pattern into sub-patterns and then decides whether a candidate pattern is frequent by joining the intermediate results obtained from searching *meta tables* OccTs_OccBits and OccCnts for every sub-pattern. Finally, OccTs+OccCandPatt, which is based on OccTs, also uses the meta table OccCandPatt to rapidly determine whether candidate patterns have ever occurred in the site. Furthermore, both OccTs and OccTs+OccCandPatt employ a *greedy* algorithm to determine the execution order of sub-patterns.

As a measure for evaluating accuracy, we used *Recall* and *Precision*. *Recall* indicates the fraction mined from all those that were actually frequent. *Precision* indicates what fraction of mined patterns are actually frequent. As shown in Fig. 5, P_{Target} denotes a set of all frequent sequential patterns in network traffic data, and P_{Mined} denotes a set of patterns mined by our method. P_{true} denotes a set of patterns belonging to both P_{Target} and P_{Mined}, P_{missed} denotes a set of patterns belonging to P_{Target} but not to P_{Mined}, and P_{false} denotes a set of patterns belonging to P_{Mined} but not to P_{Target}. With this notation, we define *Recall* and *Precision* as follows.

$$Recall = \frac{|P_{\text{true}}|}{|P_{\text{Target}}|}$$

$$Precision = \frac{|P_{\text{true}}|}{|P_{\text{Mined}}|}$$

As a performance measure, we used the average elapsed times in mining a frequent sequential pattern of the maximum length 6. In subsequent experiments, we set probability p in *retention replacement* to 1 in order to fairly evaluate the average elapsed times of all the methods.

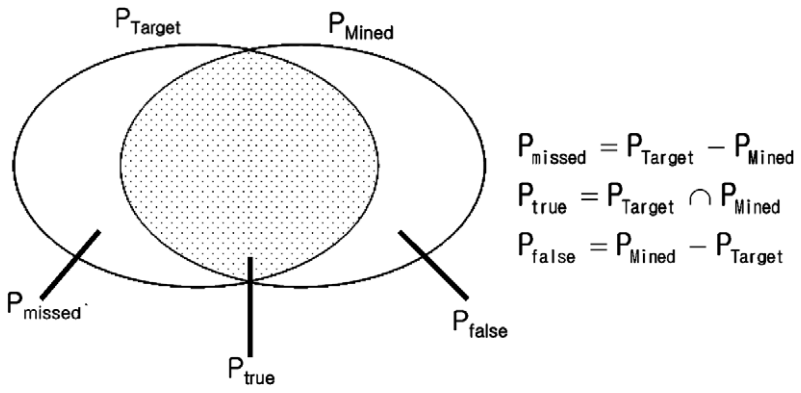


Fig. 5. A set of patterns mined and a set of target patterns.

The hardware platform is the Pentium IV 3.0 GHz PC equipped with 512 Mbytes main memory and 80 Gbytes hard disk of 7200 RPM. The software platform is the Windows XP and the Java 2 Runtime Environment 1.4.2.

5.2. Parameter settings

5.2.1. Minimum support

As *MinSup*, the minimum support, increases, the number of frequent patterns decreases rapidly, and thereby the elapsed time decreases. Fig. 6 shows the average elapsed time and the number of frequent patterns in performing experiments by the three methods by changing *MinSup* for 10 sites that store 1000 network traffic data. Here, we set the maximum time interval *MaxGap* to 20. The result of this experiment shows that, in all three methods, the average elapsed time decreases as *MinSup* increases.

In particular, with Naive, the number of frequent patterns extracted is almost proportional to the elapsed time because *Naive* determines whether a pattern is frequent or not by accessing the original data itself. Also, the elapsed time of the proposed OccTs+OccCandPatt is larger than that of OccTs when *MinSup* is 0.1. A pattern occurring even in only one site also could be also regarded as frequent and is stored in a meta table OccCanPatt. As done previously, all patterns appearing in each site are stored in the meta table OccCanPatt. Thus, the size of the meta table becomes excessively large. However, when *MinSup* is more than 0.2, the proposed OccTs+OccCandPatt performs 1.47 to 2.97 times better than Naive and 1.07 to 1.09 times better than OccTs. Thus, we set *MinSup* to 0.2 as basic value in subsequent experiments.

5.2.2. Maximum time interval

As the maximum time interval, *MaxGap*, increases, the elapsed time becomes larger since the number of patterns belonging to the time interval increases. Fig. 7 shows the elapsed time and the number of frequent patterns for experiments with different *MaxGap* for 10 sites, each of which stores 1000 network traffic data. Here, we set *MinSup* to 0.2.

The results show that the elapsed time of the proposed OccTs and OccTs+OccCandPatt is somewhat larger than that of Naive when *MapGap* is 0. With a tiny *MaxGap*, the number of frequent patterns is small. In this case, time is mainly spent in constructing the meta table OccTs rather than in finding frequent patterns.

However, as *MaxGap* increases, OccTs and OccTs+OccCandPatt perform better than Naive in proportion to the number of frequent patterns. More specifically, OccTs+OccCandPatt outperforms Naive by 1.45–2.39 times, and outperforms OccTs 1.02–1.09 times. We set *MaxGap* to 20 as a basic value for subsequent experiments.

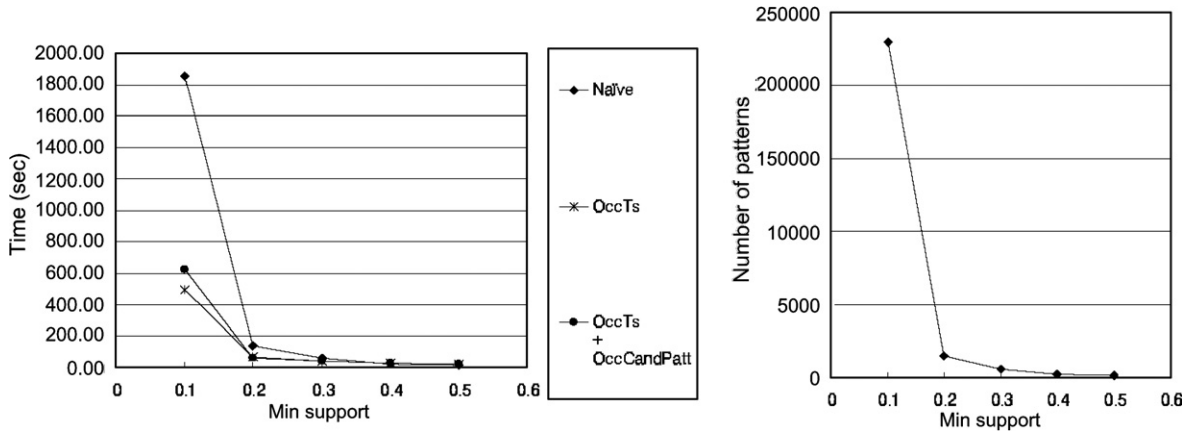


Fig. 6. The elapsed time and the number of frequent patterns with different minimum supports.

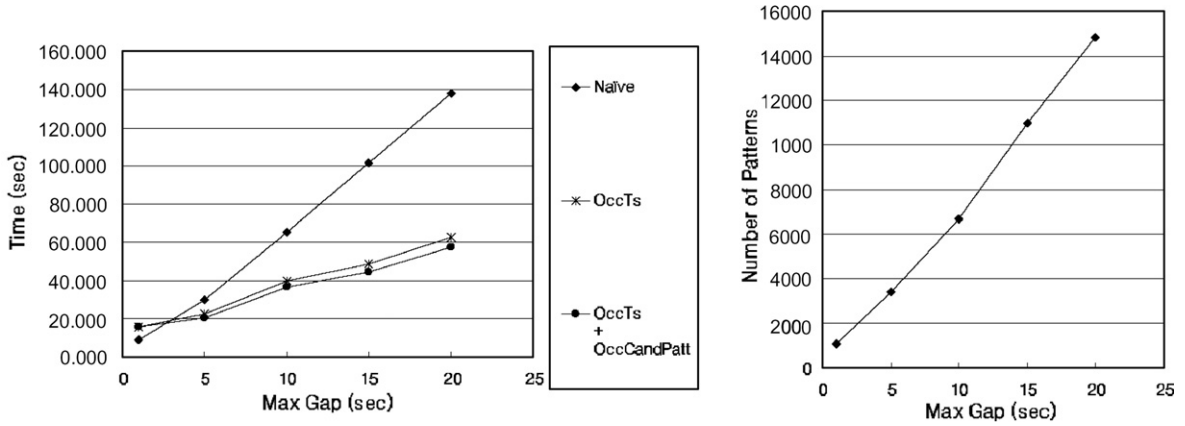


Fig. 7. The elapsed time and the number of frequent patterns with different maximum time intervals.

5.2.3. Numbers of sites and servers

The methods proposed in this paper are all applicable to the parallel processing environment. Fig. 8 shows the elapsed time with different numbers of sites and servers. Each site stored 1000 network traffic data, and thus 10 sites stored 10,000 network traffic data in total. In the cases of 20, 30, 40, and 50 sites, we stored 20,000, 30,000, 40,000, and 50,000 traffic data, respectively, in total. We set *MinSup* to 0.2 and *MaxGap* to 20.

The results show that all the three methods are hardly affected by the numbers of sites and servers. Because mining is performed on the same traffic data, the number of frequent patterns is identical even with different numbers of sites and servers. The results reveal that OccTs+OccCandPatt performs 2.34 and 1.06 times better than Naive and OccTs, respectively. We set the numbers of sites and servers to 10 and 5, respectively, in the following experiments.

5.3. Analysis of accuracy

In order to evaluate the accuracy of the proposed *N*-repository server model, we compared *Recall* and *Precision* of OccTs and Naive. Because the accuracy of both OccTs and OccTs+OccCandPatt is the same, we show only Naive and OccTs. In this experiment, we set *MinSup*, *MaxGap*, and the number of servers to 0.2, 20, and 5, respectively, as described in Section 5.2.

Firstly, we evaluated *Recall* and *Precision* with different numbers of sites. Here, we set probability p to 0.9. Fig. 9 shows the results.

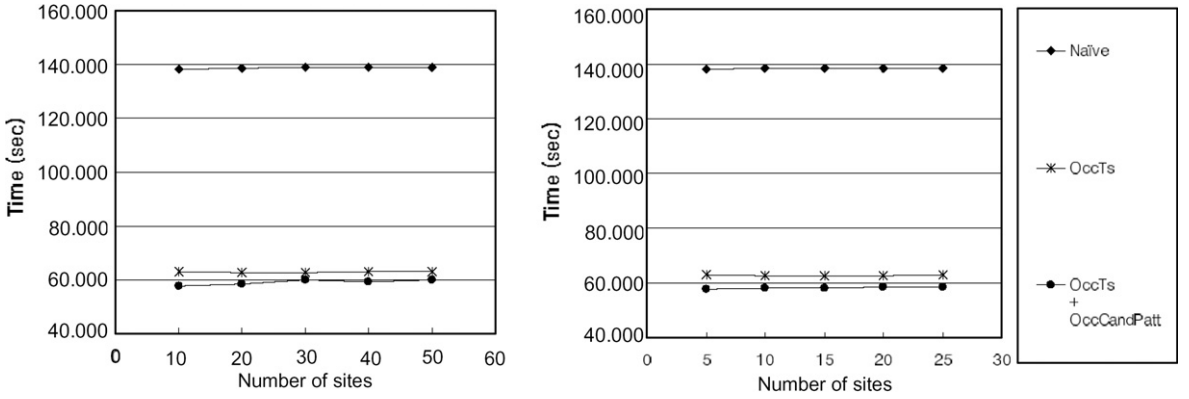


Fig. 8. The elapsed time with different numbers of sites and servers.

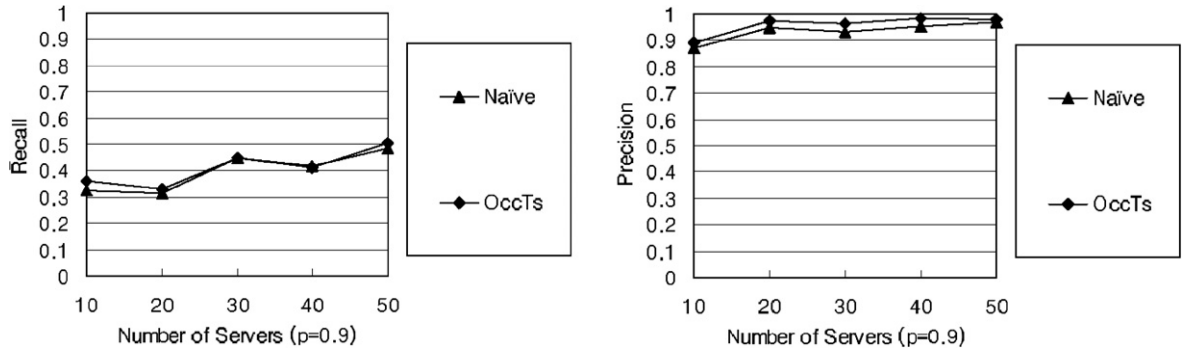


Fig. 9. The *Recall* and *Precision* with different numbers of sites.

We see that in both Naive and OccTs *Recall* and *Precision* all increase as the number of sites increases. Both methods employ the *retention replacement* to find sequential patterns whose length is longer than 2. With *retention replacement*, the greater the number of sites, the higher the accuracy. OccTs outperforms Naive 1.01–1.04 times in *Recall* and 1.01–1.32 times in *Precision*. This is because the *N*-repository server model used in OccTs enables the exact identification of all frequent patterns.

Next, we examine *Recall* and *Precision* with a different probability *p*. We set the number of sites to 50. Fig. 10 shows the results with *p* set from 0.51 to 1. We note that the *retention replacement* is inapplicable with a *p* of 0.5 [25].

The results show that in Naive and OccTs both *Recall* and *Precision* increase as *p* gets nears 1. This is due to the *retention replacement* used in both methods to find frequent sequential patterns whose length is longer than 1. OccTs performs 1.04–1.20 and 1.01–1.12 times better than Naive in *Recall* and *Precision*, respectively.

Naive is inapplicable for analyzing real Internet traffic data because it must know all items that are likely to occur in advance. Furthermore, in the above two experiments, OccTs showed accuracy higher than Naive.

5.4. Analysis of performance

In order to evaluate the performance of OccTs and OccTs+OccCandPatt, we compared them with Naive in terms of the elapsed time for mining. In this experiment, we set *MinSup*, *MaxGap*, the number of sites, and the number of servers to 0.2, 20, 10, and 5, respectively, by considering the experimental results in Section 5.2.

We measured the elapsed time with different numbers of traffic data in each site. Fig. 11 shows the results.

In all three methods, we observe that as the volume of traffic data increases, the elapsed time increases. This is because more frequent patterns appear with a larger volume of traffic data. OccTs performed 1.60–2.38

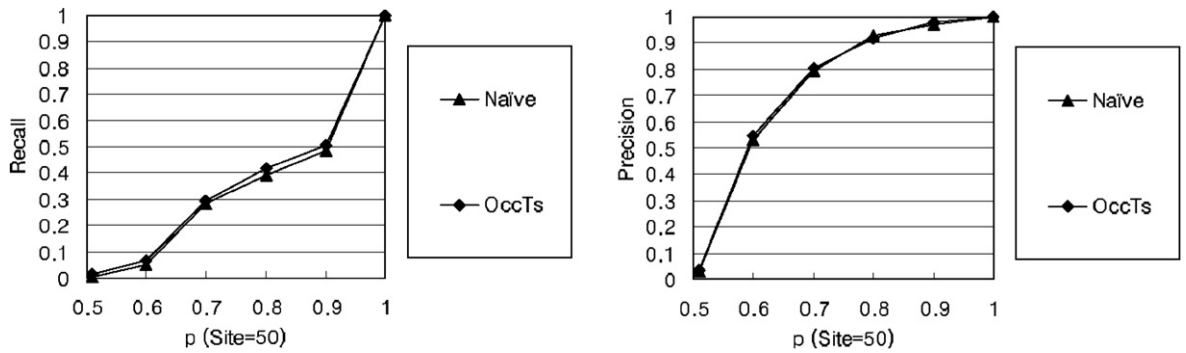


Fig. 10. *Recall* and *Precision* with different probability *p*.

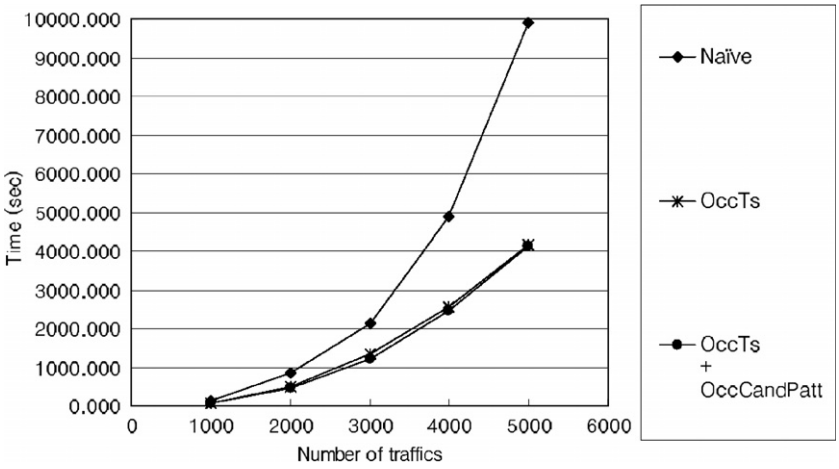


Fig. 11. The elapsed time with different numbers of traffic data.

times better than Naive. It stores all pairs of frequent items that occur within *MaxGap* into the meta table OccTs_OccBits and quickly determines whether candidate patterns occur by joining these pairs without accessing the network traffic data.

OccTs+OccCandPatt ran 1.01–1.10 times faster than OccTs. By referring to OccCandPatt, the method examines whether candidate patterns have ever occurred in the site before searching OccTs_OccBits. Therefore, it achieves a pruning effect in the mining process. That is, the total elapsed time decreases because the number of candidate patterns to be searched in OccTs_OccBits decreases. Fig. 12 shows the pruning effect obtained by using OccCandPatt. As the volume of traffic data increases, the pruning effect grows considerably. The pruning effect of OccCandPatt is 77–81% bigger than that of OccTs.

Table 4 shows an example of sequential patterns discovered in our experiment for network traffic data. The second pattern implies that printing was usually done after receiving an E-mail. Similarly, the fourth pattern implies that a dialogue via a messenger is common after web surfing.

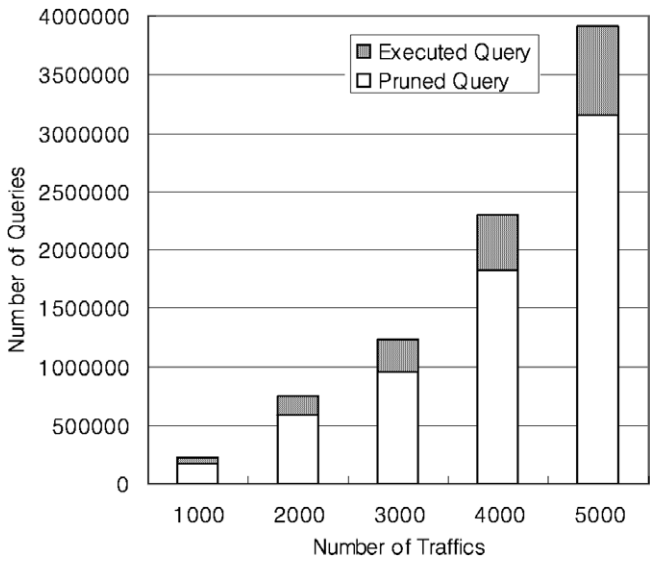


Fig. 12. Pruning effect of using OccCandPatt.

Table 4
Example of sequential patterns discovered in the experiment

Sequence	Sequential pattern
1	In, E-mail server → out, E-mail server
2	In, E-mail server → out, Printer server
3	In, E-mail server → out, Messenger
4	In, Web server → out, Messenger → in, Messenger
5	In, Web server → out, E-mail server → in, E-mail server → out, Web server → in, Messenger

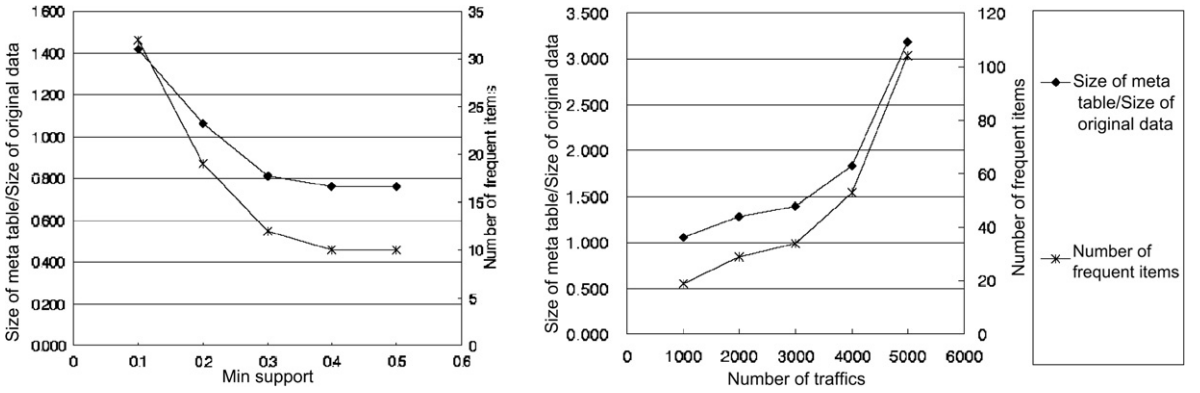


Fig. 13. The size of *meta tables*.

5.5. Size of meta tables

To evaluate the extra storage overhead of OccTs, we measured the ratio of the size of *meta tables* to that of the original data. In this experiment, we set *MinSup*, *MaxGap*, the numbers of sites, and the numbers of servers to 0.2, 20, 10, and 5 by considering the results shown in Section 5.2.

The most important factors considered in determining the size of the *meta tables* are the number of frequent items and the frequency of each frequent item in a site. The number of frequent items is affected by *MinSup* and the volume of traffic data. Fig. 13 shows the size of the *meta tables* compared with that of the original data with respect to changing *MinSup* and the number of traffic data. In the left graph, the size of the *meta tables* is affected by the number of frequent items, thereby becoming 0.76–1.42 times that of the original data. In the right graph, we see that the frequency of each frequent item and the number of frequent items increase together and that the effect is bigger than *MinSup*. The ratio of the size of the *meta tables* to that of the original data changes 1.06–3.19.

6. Conclusions and further study

In this paper, we have proposed a practical method for sequential pattern mining on network traffic data. The proposed method preserves privacy of sites and provides high accuracy of mining results. The contributions of the paper are summarized as follows: First, we have proposed a privacy preserving method that mines frequent sequential patterns from network traffic data. Our method uses the *N-repository server model* that operates as a single mining server and also employs the *retention replacement* technique that changes the answer by a given probability. Second, we have designed *meta tables* maintained in each site so as to quickly determine whether candidate patterns ever occurred in the site. Third, we have demonstrated the correctness and the efficiency of the proposed method via extensive experimentation with real-world network traffic data.

With the proposed method, we can discriminate the intruded state from the normal state by analyzing network traffic data. This makes it possible to identify sequential patterns that frequently occur only with intrusions, thereby helping to prevent intrusions. In particular, internet worms can affect intrusion traffic of the

same pattern over a large number of infected personal computers. Thus, the proposed method would be fairly useful in detecting these kinds of intrusions automatically.

The proposed method is applicable to mining sequential visiting patterns of web pages that occur frequently. The results can be used in the pre-fetching of web pages and load balancing of web servers. By using frequent sequential visiting patterns of web pages, the server can predict web pages to be accessed together and thus pre-fetch those pages to reduce their access time. Also, by distributing those web pages into multiple web servers, the servers process information more quickly due to the load balancing effect.

In bioinformatics, there have been many attempts to derive useful biomedical information, such as physical examinations results, laboratory results, disease histories, and medication records. Such information is extracted by joining the results of queries issued in distributed heterogeneous platforms. Recently, there have been many active research efforts in performing such combinations efficiently while preserving patient privacy [11,26]. The method proposed here is beneficial to this application.

In a future study using bio-medical information obtained from a heterogeneously distributed network environment, we will derive sequential patterns in disease progression, specific drug responses, disease prognoses, and gene expression profiles. We will also develop a cooperative bio-medical system that is capable of assisting doctors' decision-making process in the prediction and diagnosis of diseases by using such information.

In addition, to increase the applicability of our proposed method, we are considering extending it to a dynamic environment where online network traffic data are reflected in the mining process in real-time.

References

[1] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proceedings of the 11th IEEE International Conference on Data Engineering, Taipei, Taiwan, 1995, pp. 3–14.
[2] R. Agrawal, R. Srikant, Privacy-preserving data mining, in: Proceedings of 2000 ACM SIGMOD International Conference on Management of Data, Dallas, Texas, 2000, pp. 439–450.
[3] R. Agrawal, R. Srikant, D. Thomas, privacy preserving OLAP, in: Proceedings of 2005 ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, 2005, pp. 251–262.
[4] J. Ayres, J. Flannick, J. Gehrke, T. Yiu, Sequential pattern mining using bitmap representation, in: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Alberta, Canada, 2002, pp. 102–111.
[5] C. Clifton, D. Marks, Security and privacy implication of data mining, in: Proceedings of 1996 ACM Workshop on Data Mining and Knowledge Discovery, Montreal, Canada, 1996, pp. 15–19.
[6] P. Dokas, L. Ertoz, V. Kumar, A. Lazarevic, J. Srivastava, P. Tan, Data mining for network intrusion detection, in: Proceedings of the NSF Workshop on Next Generation Data Mining, Baltimore, Maryland, 2002, pp. 73–81.
[7] A. Evfimievski, R. Srikant, R. Agrawal, J. Gehrke, Privacy preserving mining of association rules, in: Proceedings of 2002 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, Canada, 2002, pp. 217–228.
[8] T. Fukasawa, J. Wang, T. Takata, M. Miyazaki, An effective distributed privacy-preserving data mining algorithm, in: Proceedings of the 5th International Conference on Intelligent Data Engineering and Automated Learning, Exeter, UK, 2004, pp. 320–325.
[9] M. Garofalakis, R. Rastogi, K. Shim, SPIRIT: sequential pattern mining with regular expression constraints, in: Proceedings of the 25th International Conference on Very Large Data Bases, Edinburgh, UK, 1999, pp. 223–234.
[10] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M. Hsu, FreeSpan: frequent pattern-projected sequential pattern mining, in: Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, Massachusetts, 2000, pp. 355–359.
[11] J. Han, How can data mining help bio-data analysis?, in: Proceedings of the 2nd ACM SIGKDD Workshop on Data Mining in Bioinformatics, Edmonton, Canada, 2002, pp. 1–2.
[12] Y. Hu, B. Panda, A data mining approach for database intrusion detection, in: Proceedings of 2004 ACM Symposium on Applied Computing, New York, NY, 2004, pp. 711–716.
[13] M. Kantarcioglu, C. Clifton, Privacy-preserving distributed mining of association rules on horizontally partitioned data, in: Proceedings of 2002 ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, Madison, Wisconsin, 2002, pp. 24–31.
[14] H. Kum, J. Pei, W. Wang, D. Duncan, ApproxMAP: approximate mining of consensus sequential patterns, in: Proceedings of the 3rd SIAM International Conference on Data Mining, San Francisco, California, 2003, pp. 311–315.
[15] W. Lee, S. Stolfo, K. Mok, A data mining framework for building intrusion detection models, in: Proceedings of the IEEE Symposium on Security and Privacy, Oakland, California, 1999, pp. 120–132.
[16] Y. Lindell, B. Pinkas, Privacy preserving data mining, in: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology, Santa Barbara, California, 2000, pp. 36–54.
[17] M. Lin, S. Lee, Improving the efficiency of interactive sequential pattern mining by incremental pattern discovery, in: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, Washington, DC, 2003, p. 68.2.

[18] Y. Liu, S. Sung, H. Xiong, A cubic-wise balance approach for privacy preservation in data cubes, Information Sciences 176 (9) (2006) 1215–1240.
[19] J. Luo, S. Bridges, Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection, International Journal of Intelligent Systems 15 (8) (2000) 687–704.
[20] F. Masseglia, F. Cathala, P. Poncelet, The PSP approach for mining sequential patterns, in: Proceedings of the 2nd European Symposium on Principle of Data Mining and Knowledge Discovery, Nantes, France, 1998, pp. 176–184.
[21] F. Masseglia, P. Poncelet, M. Teisseire, Incremental mining of sequential patterns in large databases, Data and Knowledge Engineering 46 (1) (2003) 97–121.
[22] J. Pei, J. Han, W. Wang, Mining sequential patterns with constraints in large databases, in: Proceedings of the 11th International Conference on Information and Knowledge Management, McLean, Virginia, 2002, pp. 18–25.
[23] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M. Hsu, Mining sequential patterns by pattern-growth: the prefixspan approach, IEEE Transactions on Knowledge and Data Engineering 16 (11) (2004) 1424–1440.
[24] B. Pinkas, Cryptographic techniques for privacy-preserving data mining, SIGKDD explorations Newsletter of the ACM Special Interest Group on Knowledge Discovery and Data Mining 4(2) (2002) 12–15.
[25] S. Rizvi, J. Haritsa, Maintaining data privacy in association rule mining, in: Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China, 2002, pp. 682–693.
[26] G. Schadow, S.J. Grannis, C.J. McDonald, Discussion paper: privacy-preserving distributed queries for a clinical case research network, in: Proceedings of the IEEE ICDM Workshop on Privacy, Security, and Data Mining, Maebashi City, Japan, 2002, Vol. 14, pp. 55–65.
[27] S. Song, Z. Huang, H. Hu, S. Jin, A sequential pattern mining algorithm for misuse intrusion detection, in: Proceedings of the International Workshop on Information Security and Survivability for Grid, Wuhan, China, 2004, pp. 458–465.
[28] R. Srikant, R. Agrawal, Mining sequential patterns: generalizations and performance improvements, in: Proceedings of the 5th International Conference on Extending Database Technology, Avignon, France, 1996, pp. 3–17.
[29] X. Yan, J. Han, R. Afshar, CloSpan: mining closed sequential patterns in large databases, in: Proceedings of the 3rd SIAM International Conference on Data Mining, San Francisco, California, 2003, pp. 166–177.
[30] J. Zhan, L. Changy, S. Matwinz, Privacy-preserving collaborative sequential pattern mining, in: Proceedings of the SIAM International Workshop on Link Analysis, Counter-terrorism, and Privacy, Lake Buena Vista, Florida, 2004. pp. 61–72.
[31] Q. Zheng, K. Xu, W. Lv, S. Ma, The algorithms of updating sequential patterns, in: Proceedings of the 5th SIAM International Workshop on High Performance Data Mining, Arlington, Virginia, 2002.
[32] S. Zhong, Privacy-preserving algorithms for distributed mining of frequent itemsets, Information Sciences 177 (2) (2007) 490–503.