Data Mining Techniques for Effective and Scalable Traffic Analysis

M. Baldi, E. Baralis, F. Risso
Dipartimento di Automatica e Informatica - Politecnico di Torino
Corso Duca degli Abruzzi, 24
10129 Torino, Italy
{mario.baldi, elena.baralis, fulvio.risso}@polito.it

Abstract

This paper describes a novel approach to traffic analysis in high speed networks based on data mining techniques. Data mining techniques are here applied as a means to effectively process the significant amount of captured data. The paper provides a first evaluation of the proposed approach in terms of its ability of extracting relevant information and its computational requirements. Such evaluation is based on experiments run on a prototypal implementation of the proposed approach.

Keywords

Traffic Analysis, Network Monitoring, Data Mining

1. Introduction

One of the most critical issues in keeping a network under control is capturing and analyzing its traffic. The complexity of these tasks is increasing as networks become faster and faster. Major problems stem from the CPU power needed to process captured network traffic and the storage requirements of historical data.

Often, traffic capturing and analysis goes through the steps depicted in Figure 1, all of which are critical when operating at high data rates. Some limited processing (e.g. associating each packet to its corresponding flow) is carried out in real-time immediately during the capture session. Then, results can be stored on a disk to be further elaborated with off-line tools, which do not suffer the limitations stemming from real-time processing.

Ad-hoc solutions based on advanced hardware (e.g. the network interface cards provided by Endace [16]) and the use of SMP workstations or even clusters can mitigate the problems related to on-line monitoring and analysis (the first steps in Figure 1). However, no straightforward solution exists to reduce the criticalities of the subsequent steps. For instance, a 10 Gbps pipe carries more than 100 TBytes in the course of a day, which is a tremendous amount of data to be stored for subsequent processing. This results in two problems: on the one hand, the infrastructure needed to store such amount of data is sophisticated and costly and, on the other hand, locating relevant information within the saved data is computationally intense and time consuming.

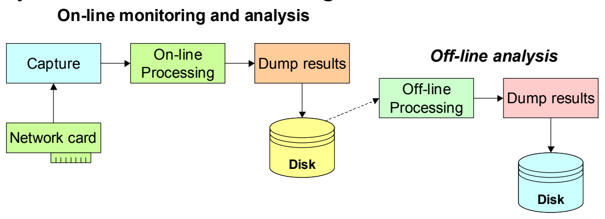


Figure 1: Basic steps in network traffic capture and analysis.

A possible solution to the first problem was presented in a previous paper [13] by the same authors and encompasses a format to represent captured packets that (*i*) limits the amount of data stored and (*ii*) enables efficient processing. The second problem is addressed by the present paper proposing the application of data mining techniques for extracting relevant information from extremely large data sets stored according to the above mentioned format.

Section 2 summarizes the work presented in [13], which is key to the solution proposed here for two reasons. First, the outcome of the network analysis process depends on the quality of the information stored during the on-line phase. Second, the complexity of off-line data processing depends on the amount and format of such data. The application of data mining techniques to traffic analysis is presented in Section 0, including some preliminary results (obtained with the prototypal implementation in the Analyzer [1] traffic capturing and analysis tool). Section 4 presents relevant literature with particular focus on work that apply data-mining techniques to network monitoring. Finally, Section 5 discusses the benefits and limitations expected from the deployment of the proposed approach, especially for what concerns the extraction of relevant information, which is still under evaluation.

2. Data Collection and Storage

Saving to disk each captured packet — or possibly just the most relevant bytes of it — may be feasible in some cases, but it anyway requires a significant amount of resources. Therefore, such approach cannot be considered as a general basis for scalable traffic monitoring procedures. In any case, single packets are not necessarily relevant for many types of traffic analysis whose focus is on packet *flows*. Our solution, similarly to the well-known and widely used Cisco System's Netflow version 9, targets such types of traffic analysis. A probe collecting data saves a given set of information related to each flow, rather than dumping to disk (part of) the content of each packet. A flow is a set of packets that have the same value in a given set of fields, which are not necessarily IP source/destination address, source/destination port, and protocol type — widely used as flow identifiers in TCP/IP networks. Our flow definition is rather general and several fields can be included in the set that best characterizes each flow. For instance, if the administrator is interested in the analysis of Differentiated Services traffic, the value of the DS field can be saved for each flow. Alternatively, if the administrator is interested only in accounting based on the IP source address, this can be the only parameter identifying a flow. Due to the flexible architecture of the underlying dumping mechanism deployed by the proposed solution, the addition of a new field in the definition of flows does not preclude the possibility of extracting statistics on previously stored data that do not have such information.

Table 1 lists the fields extracted by default from each packet by the NetLogger module [14] that is a prototypal implementation of the traffic analysis solution integrated into Analyzer. Flow identification is not necessarily based on all of these fields, some of which might even not be present; for example, ARP-related fields will not be present when analyzing IPv6 packets. The NetLogger module can be customized to extract any field in any protocol header.

Protocol	Field name
Ethernet	Source and destination address, Protocol type
VLAN	Priority, VLAN ID
ARP	Source and destination IP address
IP / IPv6	Traffic class, Prot.l type / Next header, Source and destination address
ICMP / ICMPv6	Type
TCP /UDP	Source and destination port

Table 1: Default list of fields extracted by the NetLogger module.

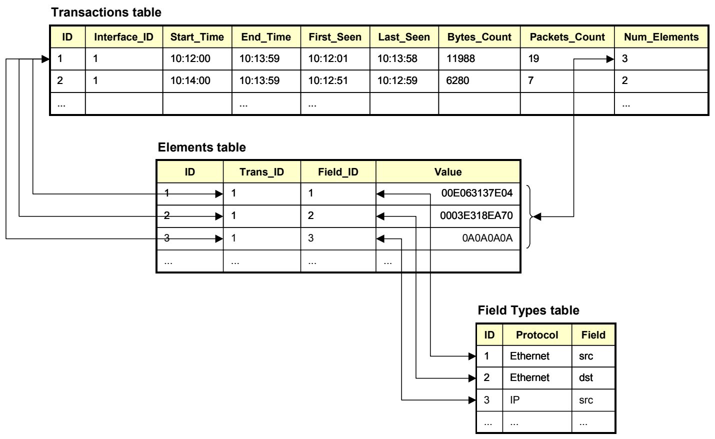
For each packet, after having devised the value of the applicable fields listed in Table 1, the probe determines the flow the packet belongs to and updates a set of counters (e.g. number of bytes/packets seen, timestamps, etc.). The selected fields are extracted for each flow and periodically dumped to disk together with the value of the above counters. The prototypal NetLogger module dumps (and clears) the entire content of the *flow cache* every N minutes; a session lasting longer will be represented by several subsequent records. The time between subsequent dumps of the flow cache is called *flushing interval*.

The NetLogger module implements the probe that captures network traffic, extracts flow information and dumps it to disk on the machine itself; therefore no protocols for exchanging data between the probe and the collector are needed. In case probe and collector are on different machines, Netflow version 9 (or the newest IETF IPFIX standard) can be used for transferring data.

To facilitate further information processing in our prototype flow information is saved into a database. SQLite was selected for this purpose since it provides very fast access and its overhead is only 5 times the time required to store data on a flat (text) file, as shown by the figures presented in [13].

Figure 2 shows with an example the structure of the database (all details can be found in [13]). Although this structure is slightly more complex than the one traditionally deployed in traffic monitoring applications (one table with a fixed number of fields, and one record per flow), it has proved to be more flexible because it can accommodate a different number fields for each flow record (called *transactions* within the database structure) without changing the database structure.

Experimental results in [13] show that the NetLogger module storing each flow record in an SQLite database at the default flushing interval of 2 minutes features a 20 to 1 reduction in disk space requirement when compared to saving each packet. The resulting disk files can be further compressed by means of general-purpose compression utilities, such as gzip, thus obtaining disk-saving factors of more than 80:1. However, the amount of data resulting from this process is still rather high. In fact, the authors believe a further reduction in data size can only be obtained through the deployment of data mining techniques, as presented in the next section.



Transactions table

ID	Interface_ID	Start_Time	End_Time	First_Seen	Last_Seen	Bytes_Count	Packets_Count	Num_Elements
1	1	10:12:00	10:13:59	10:12:01	10:13:58	11988	19	3
2	1	10:14:00	10:13:59	10:12:51	10:12:59	6280	7	2
...			...	...				...

Elements table

ID	Trans_ID	Field_ID	Value
1	1	1	00E063137E04
2	1	2	0003E318EA70
3	1	3	0A0A0A0A
...	...	...	...

Field Types table

ID	Protocol	Field
1	Ethernet	src
2	Ethernet	dst
3	IP	src
...	...	...

Figure 2. Snapshot of records stored in the database.

3. Extracting relevant information

A set of standard statistics (e.g. the protocol distribution, the amount of traffic sent by every host, etc.) can be easily obtained from the data stored as described above. However, even though the proposed approach results in significantly less information than a raw packet dump would produce, locating added-value information (e.g., locating an ongoing security attack) might be extremely cumbersome, if at all possible, for the network administrator.

We have been experimenting the application of data mining techniques to large databases structured as described in the previous section, wherein each sample of a flow (called "transaction" in the NetLogger database) is represented by a variable-length record. Particularly, our prototypal NetMiner module [15], integrated in Analyzer, implements data mining techniques for the extraction of Frequent Itemsets and Association Rules [2].

An *Itemset* is a set of elements — pairs of field/value records in the database — characterized by a given value in one or more fields (e.g IP source address and TCP source port). An Itemset is considered *Frequent* if its cardinality exceeds a given threshold with respect to the total number of samples.

Two parameters are used to characterize frequent itemsets: *minimum support* and *maximum number of items* to be considered within a transaction. The support of an itemset X (e.g. {`dest_host=X`, `dest_port=Y`}) is defined as the number of transactions where X is present, divided by the total number of transactions. The data mining process returns only itemsets whose support is greater than a given threshold named minimum support. This aims at avoiding returning itemsets with little relevance. The itemset {`dest_host=X`, `dest_port=Y`} has *cardinality* two; a high value for the maximum number of items (i.e., itemset cardinality) makes processing heavier, but allows discovering more complex relationships that link together several elements.

Association Rules are extracted from frequent itemsets and show correlations among (contained) elements. Usually, association rules are written in the form:

```
A => B (support %, confidence %)
```

A is often referred to as the *body* of the rule, while B as the *head* of the rule. Parameters used to customize the quantity (and quality) of data mining results are *minimum support*, *minimum confidence*, and *maximum number of items* to be considered within a transaction. The support is the number of transactions in which the body of the association rule (e.g. {`IP dest_addr=S`}) is present, divided by the total number of transactions. The confidence is the support of sets that contain body and head (A and B), divided by the support of sets that contain the body (A). The association rule holds if its confidence is above a given minimum level.

For instance, if a host S is active mostly as a web server, the association rule

```
If (IP dest_addr = S) → TCP dest_port = 80 (1.2%, 87%)
```

shows that there is a high probability (87%) that the flows directed to the server (characterized by the value S in the IP destination address field) contain 80 (the default TCP port for a web server) in the TCP destination port field, and that this association rule is valid for the 1.2% of the transactions.

3.1. Extracting and Understanding Mined Data

Data stored in the database by the NetLogger module are exported to a flat file, then processed by the Apriori [3] tool, which is widely used for extracting frequent itemsets from a database. Frequent itemsets and association rules are derived according to the parameters chosen by the user (confidence, support, maximum number of items returned within each itemset) and the results are stored in a new SQLite database. The structure of the new database (shown in Figure 3) has been designed in order to be flexible and support any type of association rule and frequent itemsets, irrespective of the number of items involved. Two examples of stored association rule and frequent itemset are shown in Figure 4 and Figure 5.

Interpretation of results of the data mining process is a major problem. This is mainly due to the large amount of information returned by data mining techniques that the network administrator is required to go through. For example, it is not uncommon that hundred of thousands association rules be identified on a traffic trace. The problem of sifting through them is emphasized by the fact that the network administrator is not — and should not become — a data mining expert.

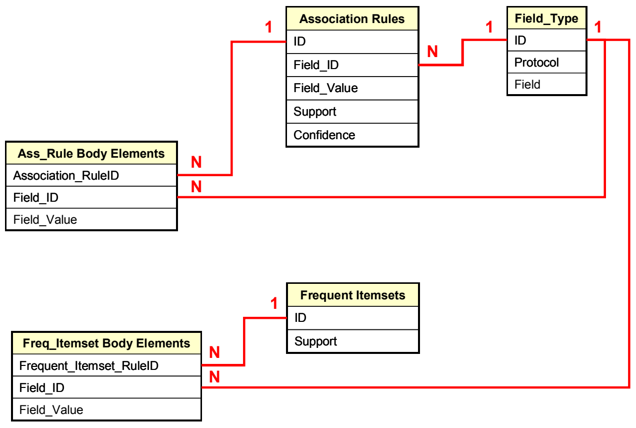
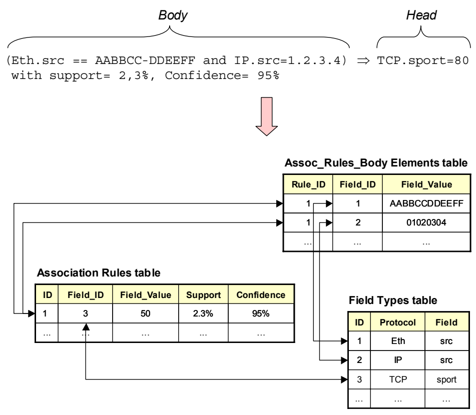


Figure 3: Structure of the database containing mined results.

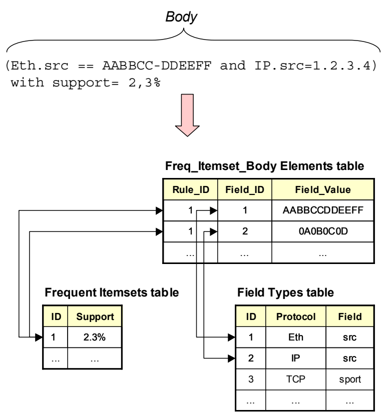


Body
Head
(Eth.src == AABBCC-DDEEFF and IP.src=1.2.3.4) ⇒ TCP.sport=80
with support= 2,3%, Confidence= 95%
Assoc_Rules_Body Elements table
Association Rules table
Field Types table

Rule_ID	Field_ID	Field_Value
1	1	AABBCCDDEEFF
1	2	01020304
...	...	...

ID	Field_ID	Field_Value	Support	Confidence
1	3	50	2.3%	95%
...	...	...	...	...

ID	Protocol	Field
1	Eth	src
2	IP	src
3	TCP	sport
...	...	...



Body
(Eth.src == AABBCC-DDEEFF and IP.src=1.2.3.4)
with support= 2,3%
Freq_Itemset_Body Elements table
Frequent Itemsets table
Field Types table

Rule_ID	Field_ID	Field_Value
1	1	AABBCCDDEEFF
1	2	0A0B0C0D
...	...	...

ID	Support
1	2.3%
...	...

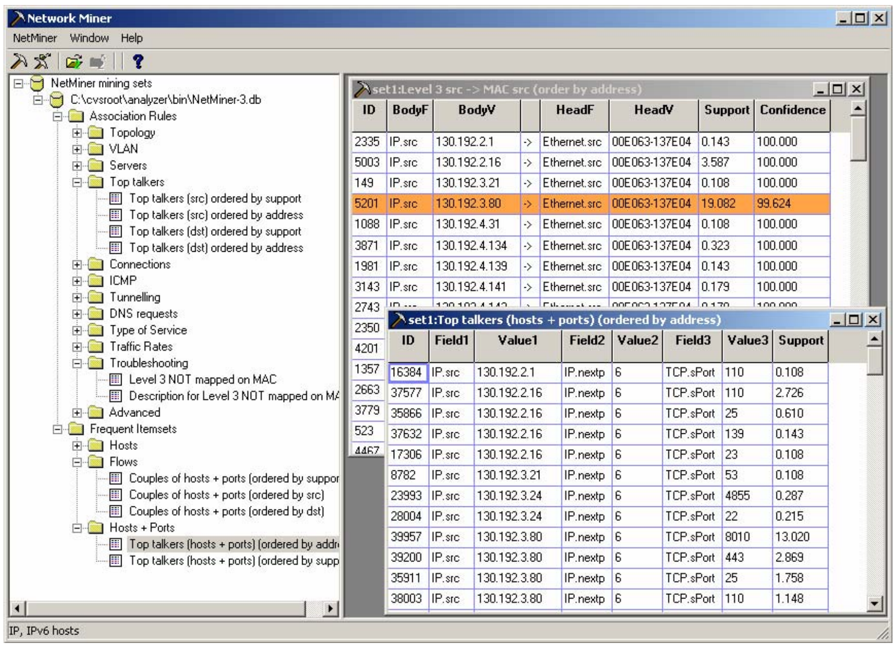
ID	Protocol	Field
1	Eth	src
2	IP	src
3	TCP	sport
...	...	...

Figure 4: Example of an Association Rule stored in the database.

Figure 5: Example of a Frequent Itemset stored in the database.

Being designed specifically for network analysis applications, the user interface of the NetMiner module (shown in Figure 6) facilitates the network administrator in browsing the results of the data mining process. In essence, the NetMiner user interface provides ways to group rules that lead to the same networking semantics and give separate access to them. For instance, the application of the data mining process on a traffic capture might yield a set of rules that associate each IP address to a MAC address, i.e. the link-layer address of the network card sending the packets. As shown in the leftmost panel in Figure 6, the NetMiner user interface allows (*i*) a name to be associated to each relevant set of rules and (*ii*) the set of rules to be organized in categories. The view in the rightmost panel in Figure 6 — obtained once the name corresponding to a set of rules is selected in the leftmost panel — is created by executing an SQL query on the database storing mined data.

The NetMiner user interface provides a list of predefined sets of rules and corresponding queries that can help network administrators getting a certain level of understanding of the mined data. Since the list of sets of rules and corresponding queries are specified in a XML file, they can be easily extended by the network manager himself or by third parties.



```
NetMiner mining sets
  C:\cvsroot\analyzer\bin\NetMiner-3.db
    Association Rules
      Topology
      VLAN
      Servers
      Top talkers
        Top talkers (src) ordered by support
        Top talkers (src) ordered by address
        Top talkers (dst) ordered by support
        Top talkers (dst) ordered by address
      Connections
      ICMP
      Tunnelling
      DNS requests
      Type of Service
      Traffic Rates
      Troubleshooting
        Level 3 NOT mapped on MAC
        Description for Level 3 NOT mapped on MA
      Advanced
    Frequent Itemsets
      Hosts
      Flows
        Couples of hosts + ports (ordered by suppor
        Couples of hosts + ports (ordered by src)
        Couples of hosts + ports (ordered by dst)
      Hosts + Ports
        Top talkers (hosts + ports) (ordered by addr
        Top talkers (hosts + ports) (ordered by supp
IP, IPv6 hosts
```

set1:Level 3 src -> MAC src (order by address)

ID	BodyF	BodyV		HeadF	HeadV	Support	Confidence
2335	IP.src	130.192.2.1	->	Ethernet.src	00E063-137E04	0.143	100.000
5003	IP.src	130.192.2.16	->	Ethernet.src	00E063-137E04	3.587	100.000
149	IP.src	130.192.3.21	->	Ethernet.src	00E063-137E04	0.108	100.000
5201	IP.src	130.192.3.80	->	Ethernet.src	00E063-137E04	19.082	99.624
1088	IP.src	130.192.4.31	->	Ethernet.src	00E063-137E04	0.108	100.000
3871	IP.src	130.192.4.134	->	Ethernet.src	00E063-137E04	0.323	100.000
1981	IP.src	130.192.4.139	->	Ethernet.src	00E063-137E04	0.143	100.000
3143	IP.src	130.192.4.141	->	Ethernet.src	00E063-137E04	0.179	100.000
2743							
2350							
4201							
1357							
2663							
3779							
523							
4467							

set1:Top talkers (hosts + ports) (ordered by address)

ID	Field1	Value1	Field2	Value2	Field3	Value3	Support
16384	IP.src	130.192.2.1	IP.nextp	6	TCP.sPort	110	0.108
37577	IP.src	130.192.2.16	IP.nextp	6	TCP.sPort	110	2.726
35866	IP.src	130.192.2.16	IP.nextp	6	TCP.sPort	25	0.610
37632	IP.src	130.192.2.16	IP.nextp	6	TCP.sPort	139	0.143
17306	IP.src	130.192.2.16	IP.nextp	6	TCP.sPort	23	0.108
8782	IP.src	130.192.3.21	IP.nextp	6	TCP.sPort	53	0.108
23993	IP.src	130.192.3.24	IP.nextp	6	TCP.sPort	4855	0.287
28004	IP.src	130.192.3.24	IP.nextp	6	TCP.sPort	22	0.215
39957	IP.src	130.192.3.80	IP.nextp	6	TCP.sPort	8010	13.020
39200	IP.src	130.192.3.80	IP.nextp	6	TCP.sPort	443	2.869
35911	IP.src	130.192.3.80	IP.nextp	6	TCP.sPort	25	1.758
38003	IP.src	130.192.3.80	IP.nextp	6	TCP.sPort	110	1.148

Figure 6: Screenshot of the NetMiner module: list of relevant sets of rules in the left panel and results of the corresponding queries in the right panel.

The queries can be designed to highlight potential network problems by singling out “strange” results among the mined data. For instance, an IP address that is being associated to more than one MAC address (which, in data mining terms, means that the association rule between the IP address and the MAC address has confidence < 100%) can be a symptom of a network misconfiguration. One of the predefined NetMiner queries on the mined data (see the background right pane of Figure 6) easily brings this to the attention of the network manager.

The NetMiner module implements also a “compare mining sets” feature: queries (as defined in the XML file mentioned above) are run on two NetMiner databases and the differences between the two sets of results are shown. The comparison of databases devised from traffic captured at different times provides the network manager with a clear view of changes. For example, selecting the query intended to single out servers in the comparative mode allows the servers activated/deactivated in the time between the two captures to be easily identified.

3.2. Interpreting Mined Data

Among the most interesting results of our tests is the ability to effectively locate peer-to-peer applications. Locating and monitoring traffic generated by these applications that are usually installed and controlled directly by network users is important for network administrators. However, this is not easy with traditional traffic monitoring and analysis methods. For example, often peer-to-peer applications are not among the top network speakers (i.e., they do not generate much traffic), thus they cannot be identified by looking for large amount of data being transferred. Moreover, they use random ports, therefore they cannot be located by looking for traffic originated from or destined to specific ports. Conversely, the proposed data mining based approach easily locates them by singling out association rules between hosts and ports used.

Even though the specific issue of identifying peer-to-peer applications could be addressed by tools keeping track of network flows (and some other heuristics) in addition to counting transmitted bytes, data mining techniques provide an additional advantage: being agnostic with respect to network data. Therefore they have the potential of identifying unanticipated anomalies by extracting new rules as traffic patterns changes. However, deploying such potential requires the network manager to have a look at the "new rules" (the ones that have not been classified by existing queries) and to figure out whether they are signs of new network anomalies.

3.3. Performance Evaluation and Scalability Issues

From Data Mining theory, the complexity of the Apriori algorithm is linear in the number of input transactions and grows exponentially in the number of attributes (i.e. the number of elements of each itemset, which represents the complexity of the relations to be extracted). However, this does not provide any hints on the real cost of this approach. Therefore, a set of experiments was conducted on the NetMiner module to quantitatively assess the *performance* and *scalability* of the proposed solution. The assessment was based on the following indexes:

- *processing time* representing the effort required to extract data-mining results from a set of real-life campus traffic captures;
- *mined information* being produced by the data mining process;
- *disk space* required to store the extracted results.

The experiments were conducted with four sets of transactions, each one devised from one hour of traffic captured in the campus network of our University. The four sets contain a significantly different number of transactions (i.e. flow records generated by the NetLogger module), since the corresponding traffic had been captured in different times of day: Set 1) 268860 transactions, captured at night; Set 2) 379695 transactions, captured in the evening, Set 3) 545273 transactions, captured in the afternoon; Set 4) 921250 transactions, captured in the morning.

3.3.1. Processing Time

Table 2 reports the average processing time per transaction for each of the abovementioned sets (rows) and for different values of minimum support (columns). As expected, smaller values of minimum support (which means that also not frequent relations are extracted) imply longer per-transaction processing time. Less obvious, per-transaction processing time decreases with growing set sizes, thus showing that the approach has good scalability. An experimental evaluation of the optimal support is left for future work.

		Minimum support						
		0.5%	**0.25%**	**0.125%**	**0.05%**	**0.025%**	**0.0125%**	**0.005%**
Dataset	**Set 1**	1.16	1.26	1.48	1.70	2.64	3.95	10.46
	Set 2	1.17	1.24	1.42	1.57	2.24	3.19	7.49
	Set 3	1.15	1.22	1.31	1.40	1.93	2.52	5.20
	Set 4	1.08	1.09	1.17	1.20	1.49	1.91	3.36

Table 2: Processing time (ms) for each transaction.

A more accurate look at the figures in Table 2 seems to suggest that scalability is better for smaller values of minimum support (compare the processing times shown in the first and last columns). However, this phenomenon is related to the way processing time was measured as the time required by NetMiner for (i) *extracting* the transactions, (ii) *converting* them into a format suitable for data-mining, (iii) *extracting* frequent itemsets / association rules, and (iv) *importing* them into the results database. Only the third step is actually related to the data mining process and its execution time is longer with smaller minimum support values. The first two steps aim at extracting data from the NetLogger database and format them in the way accepted by Apriori and their cost grows linearly with the number of transactions. The last step inserts the Apriori results into another database in order to make access to them easier and grows linearly with the number of extracted relations. With large minimum support values, step (iii) does not require long execution time, hence the constant per-transaction processing time of steps (i) and (ii) is predominant. Conversely, the longer execution time of step (iii) weighs more on the overall per-transaction processing time with low values of minimum support. Steps (i), (ii), and (iv) are strictly related to our implementation and can be avoided.

3.3.2. Number of Data-Mining Results

Table 3 shows the number of association rules extracted from the four sets of transactions (rows) with different values of minimum support (columns). The fact that the total number of extracted rules decreases for bigger sets, is a further manifestation of the scalability of the data mining techniques deployed in the presented solution.

		Minimum support						
		0.5%	0.25%	0.125%	0.05%	0.025%	0.0125%	0.005%
Dataset	Set 1	22892	52589	106949	242110	430759	860685	2987204
	Set 2	22255	46917	96222	250953	456386	918005	2944277
	Set 3	19274	44415	84887	230933	461037	894737	2562332
	Set 4	16678	38964	85821	206277	433320	868044	2257974

Table 3: Number of association rules extracted by the mining process.

3.3.3. Size of the Data-Mining Result Sets

The third set of measures concerns the size of the resulting NetMiner database for different transaction sets (rows) and different values of minimum support (columns). This data can help in deciding whether saving mining results rather than the set of transactions devised from the captured traffic, represents an efficient solution for long-term storage to enable historical analysis of traffic.

		Minimum support						
		0.5%	0.25%	0.125%	0.05%	0.025%	0.0125%	0.005%
Dataset	Set 1	6.8	15.5	31.9	73.1	132.5	266.5	933.1
	Set 2	6.6	13.8	28.6	76.2	139.0	283.0	919.6
	Set 3	5.6	13.1	25.0	69.9	141.7	276.1	798.0
	Set 4	4.8	11.5	25.5	62.2	132.2	267.2	699.3

Table 4: Size (MB) of the results database.

Table 4 and Table 5 confirm the scalability properties previously shown. In this case, not only the per-transaction space needed to save information decreases (Table 5) with the set size (i.e., the amount of traffic captured), but also the total size of the results database is smaller (Table 4). If the minimum support is above a certain limit (0.0125 in our experiments), the size of the results database is smaller than the size of the original transaction database and the set of extracted association rules can be considered a sort of "compressed description" of the original transaction set. .

		Minimum support						
		0.5%	0.25%	0.125%	0.05%	0.025%	0.0125%	0.005%
Dataset	Set 1	25	58	119	272	493	991	3471
	Set 2	17	36	75	201	366	745	2422
	Set 3	10	24	46	128	260	506	1463
	Set 4	5	12	28	68	144	290	759

Table 5: Average per-transaction size (Bytes) of the results database.

3.3.4. Data Mining and Scalability Issues

The scalability of the mining process with respect to the size of the transaction set can be explained as follows. First, many of the association rules are extracted in any case, irrespective of the number of transactions contained in the data set. For instance, a rule that associates an IP address with the corresponding MAC address will always be extracted, be the database small or large. Consequently, while these association rules represent a high per-transaction contribution (in terms of all of the above mentioned metrics) when dealing with small transaction sets, their per-transaction contribution becomes less relevant as the set size grows. Second, when dealing with a larger number of transactions some rules may not be extracted because their support falls below the minimum support threshold.

In other words, large transaction sets are prone to providing a higher number (in relative — per-transaction — terms) of high-significance rules, decreasing the impact of low-significance ones.

4. Related Work

While [4] presents some theoretical considerations on the deployment of data mining techniques to network monitoring, to the best of the authors' knowledge, experiences and results are not available in the literature. Experiences with the application of data mining techniques to network related problems, such as web log analysis [5] and enterprise-wide management [6], can only be found.

The application of data mining techniques to network traffic is more widely studied and deployed in the realm of network security, particularly *intrusion detection*. Historically, IDS have been built using a signature-based approach, also referred as *misuse detection*: detection methods are based on identifying "signatures" present inside malicious traffic; where the "signature" can be a sequence of bytes, in a packet or a sequence of packets. Misuse detection techniques, that are widely adopted, provide an efficient way to identify known problems, for which a signature exists. For example, an open-source IDS system like `snort` [7] can reach 100% detection rate on a test set made up of only known intrusions. Additionally, these systems have a very low *false positive rate* (i.e. the percentage of normal traffic reported to be an intrusion). However, these techniques are, in general, not effective against novel attacks that have no matched rules or pattern yet and require regular updates with signatures, that developers have to manually program, of newly discovered intrusions.

Data mining algorithms are being used for research purposes and in some commercial products to build a model of the network traffic and to search for patterns that deviate significantly from the established normal usage profiles: these are "anomalies" and can be treated as possible intrusions. *Anomaly detection techniques* can be effective against unknown or novel attacks since they do not require knowledge about specific intrusions. Also, they do not require manual programming (such as description of signatures of new intrusions), since the automatically inferred normal behavior of the network is used to identify attacks.

The first applications of data mining to IDSs deployed *labeled data* to train the system [8] [9]: a training set, consisting of traffic already marked as "normal" or "intrusion", is supplied to a learning algorithm. The system is then provided with unlabeled traffic and classifies it as normal or containing an intrusion. Thus, the initial application of data mining techniques to IDSs in fact relates to “misuse” detection rather than anomaly detection. The advantage of these techniques over standard “signature-based” methods is limited by the fact that manually labeling a set to train the algorithm is equivalent to defining signatures. In addition, some of the experiments showed that the system is able to identify only intrusions already present into the training set.

Instead, data mining methods for intrusion detection working with unlabeled traffic provide significant advantage over signature-based methods. LERAD [10] discovers relationships among attributes with the objective of modeling network protocols. Results are encouraging (>60% of detection rate, with a very low false positive rate), but far from making the algorithm usable in real-life scenarios.

Other approaches [11][12] use clustering-based techniques to detect anomalies, aiming at identifying “normal values” for certain fields in packet headers versus anomalies. This method showed a detection rate around 40%-55% with a 1.3%-2.3% false positive rate, which is still unacceptable.

5. Concluding Remarks

The object of this work is the application of data mining techniques to network monitoring, particularly for keeping track of active hosts and some of the characteristics of their network traffic. Several work propose the deployment of data-mining techniques within Intrusion Detection Systems (IDS), but according to their results such approaches are still far from being applicable in a real network. We propose a more limited, but feasible, approach.

The mining process (therefore the extracted results) is identical for both IDS and network monitoring applications. However, our methodology does not try to infer IDS properties from the extracted rules, but focuses on a subset of rules that help describing the network and its traffic: which nodes are servers, which are routers, which are the top talkers (in terms of bytes/sec and sessions/sec), and so on. A network administrator benefits from this data because he can identify unauthorized servers (e.g., peer-to-peer server nodes), network anomalies (e.g., IP spoofing activity), and even security threats (e.g. a large number of connections to a remote server due to a Trojan that infected some local machines).

While the data collection and storage approach proposed in Section 2 and its implementation in the NetLogger module can be considered stable, the data mining approach and its implementation within the NetMiner module still need detailed evaluation and a field trial. Even though the tests conducted on the campus network of our University (including about 6,000 end-systems) enabled us to gain some insight in the benefits and shortcomings of the proposed approach, much more can be learned through a more extensive deployment in various environments and by various users.

This preliminary work using NetMiner has demonstrated that the approach features lights and shadows. Data mining techniques have shown good scalability, which is perhaps the most challenging issue in high speed network monitoring. However, the interpretation of results of the data mining process is far from being straightforward, mainly due to the large amount of information that the network administrator is required to go through. Our solution to this problem consisted in the creation of a graphic user interface (GUI) designed to hide the above complexity. A drawback of the deployment of such GUI it that it might as well hide relevant results that are not being classified within predefined categories. We believe that more work concerning this point is required.

An interesting evolution of our approach is to make it able to work in real-time. Currently, the NetMiner module gets data from the NetLogger module, processes it and returns the results. While this works reasonably well to signal network anomalies (such as an abusive server), it is not suitable for real-time network monitoring. A real-time engine that processes traffic according to the rules that have been defined as "interesting" (i.e. the rules presented by the above-mentioned GUI) might open the way to new applications. However, in order to be able to process network data in real-time a program has to be created that looks for data matching "interesting" rules. Such engine will loose the ability to discover unknown relationships because it does no longer implement a data mining algorithm — that is able to automatically discover new relationships (e.g. a relationship between a specific IP source addresses and TCP ports).

A network administrator that simply uses the GUI without digging manually into the data mining results will incur exactly the same problem. An intermediate approach could be the deployment of a run-time engine in conjunction with an offline data-mining process whose objective is to extract unknown results, which can be used to discover anomalies.

More investigation and experimental results are needed on an important by-product of the proposed approach: using the outcome of the data mining process as an extremely compact representation of the captured network traffic. In fact, the size of the output of the proposed data mining techniques can be more than 50 times smaller than size of the database generated by the network traffic storage solution presented in Section 2 (which is already several times smaller than a raw network traffic dump). Consequently, it would be interesting to assess the amount of the relevant information that can be inferred from the output of the data mining process versus the one that can be inferred from the traffic database (as described in Section 2). In this case, the latter could be discarded and only the former kept for later reference, thus dramatically reducing the storage requirement for keeping historical network traffic traces.

Future work will include also the evaluation of clustering techniques in addition to the already deployed frequent itemsets and association rules. Moreover, it should be interesting to explore the possibility to add temporal capabilities that would enable the correlation of sequential events.

ACKNOWLEDGMENTS

The authors wish to thank A. Cerutti and P. Giverso for their work on the implementation of the NetLogger and NetMiner modules and P. Fasano who once suggested us to use data mining techniques for analyzing network traffic.

References

[1] The NetGroup at Politecnico di Torino, *Analyzer 3.0*, http://analyzer.polito.it/

[2] J. Han, M. Kamber, *Data mining: concepts and techniques*, Morgan Kaufmann, 2001.

[3] R. Agrawal & R. Srikant, *Fast Algorithms for Mining Association Rules*, Proceedings of the 20th Int.l Conference of Very Large Data Bases, 1994.

[4] K.E. Burn-Thornton, J. Garibaldi, A.E. Mahdi, *Pro-active network management using data mining*, Proceedings of the Global Telecommunications Conference, 1998 (GLOBECOM 98), November 1998.

[5] Q. Yang, H.H. Zhang, *Web-log mining for predictive Web caching*, IEEE Transactions on Knowledge and Data Engineering, Volume: 15, Issue: 4, July-Aug. 2003, pg. 1050 – 1053.

[6] A. Knobbe, D. Van der Wallen, L. Lewis, *Experiments with data mining in enterprise management*, Proceedings of the 6th IFIP/IEEE Int.l Symposium on Integrated Network Management, 24-28 May 1999, pg 353 – 366.

[7] Martin Roesch, *Lightweight Intrusion Detection for Networks*, Proceedings of the 13th Conference on Systems Administration (LISA 99), pg. 229 – 238.

[8] W. Lee, S.J. Stolfo, *Data mining approaches for intrusion detection*, Proceedings of the 7th USENIX Security Symposium, 1998.

[9] W. Lee, S.J. Stolfo, *A Framework for Constructing Features and Models for Intrusion Detection Systems*, ACM Transactions on Information and System Security, vol. 3, November 2000.

[10] M.V. Mahoney, P.K. Chan, *Learning rules for anomaly detection of hostile network traffic*, 3rd IEEE International Conference on Data Mining 2003 (ICDM 2003), November 2003, pg. 601-604.

[11] Karlton Sequeira, Mohammed J. Zaki, *ADMIT: Anomaly-base Data Mining for Intrusions*, 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 2002.

[12] L. Portnoy, E. Eskin, S.J. Stolfo, *Intrusion detection with unlabeled data using clustering*, Proceedings of the ACM Workshop on Data Mining Applied to Security (DMSA 2001), November 2001.

[13] M. Baldi, E. Baralis, F. Risso, *Data Mining Techniques for Effective Flow-based Analysis of Multi-Gigabit Network Traffic*, Proceedings of the IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCom 2004), Split (Croatia), October 2004, pg. 330 – 334.

[14] A. Cerutti, *Ingegnerizzazione di una architettura distribuita per l'analisi di traffico di rete*, Laurea Degree thesis, Politecnico di Torino, September 2003.

[15] P. Giverso, *Data Mining Techniques for Network Traffic Analysis*, Laurea Degree thesis, Politecnico di Torino, September 2003.

[16] Endace Measurement Systems, web site at http://www.endace.com