TRAFFIC ANALYSIS IN HIGH SPEED NETWORK USING DATA MINING TECHNIQUES

M.LAKSHMIPRIYA.,(M.E),G.LOGANATHAN.,M.E.,(AP/IT)
Dept of CSE MUTHAYAMMAL ENGINEERING COLLEGE
lpmlakshmipriya@gmail.com,logo.mscme@gmail.com

Abstract:

This paper describes an effective way to capture network traffic in high speed networks and analyze the captured traffic using data mining techniques. We address the problems in storing and processing large amount of data in this paper. After having provided the current state of the art in significantly reducing the amount of data stored we describe our approach, which consists of two strictly related parts. It defines a format to represent captured packets that (i) limits the amount of data stored and (ii) enables efficient processing. Data mining techniques widely studied and deployed for extracting relevant information from extremely large data bases are applied to our problem. A prototypal implementation of the proposed approach has been integrated into the Analyzer traffic capturing and analysis tool. It 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.

INTRODUCTION:

The most critical issue in keeping a widespread network under our control is capturing and analyzing its traffic. The complexity of the task increases as the network becomes faster and faster. Traffic capturing and analysis goes through the steps depicted in the following figure-Figure 1, all of which are critical when operating at high data rates.

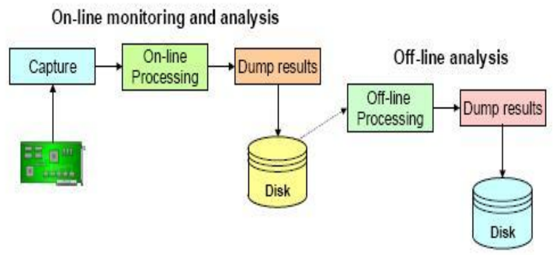


Figure 1 - Basic steps in network traffic capture and analysis

Some equipment vendors, such as Endace offer network interfaces specifically designed for supporting packet capture at high data rates (e.g., 10 Gbps), thereby facilitating the realization of the first step in Figure 1.The time required to receive a minimum size Ethernet frame at 10 Gbps speed is less than 70 ns, which leaves a few hundred clock cycles to a multi-GHz processor for handling a captured packet. This makes the realization of the second step critical. However, the deployment of multi-processor machines that concurrently process multiple packets increases the time available for handling each packet.

While ad-hoc solutions based on advanced hardware can mitigate the problems related to the first two steps in Figure 1, no straightforward solution exists to reduce the criticalities of the next steps and we need to face the below two problems: The infrastructure needed to store such amount of data is sophisticated and costly

Locating relevant information within the saved data is computationally intense and time consuming.

Existing Methods to reduce the amount of stored information:

Two methods are known in literature to reduce the amount of information about network traffic to be stored before further processing.

Packet Sampling

We'll be capturing only a subset of the packets, e.g. one out of N. Although several studies demonstrate that statistical properties can be inferred from sampled traffic without any noticeable loss of information, this approach is not effective when all packets must be analyzed. One example is the detection of network attacks that are usually based on a small number of packets exploiting a security bug (e.g. ping of death)

Flow Extraction

The second method is based on the fact that each packet can be associated to a flow (e.g. a TCP connection). A flow can be defined as the set of packets that share the value of some fields in their headers (e.g. IP source and destination addresses, TCP source and destination ports, etc.), which can be seen as the signature of the flow. Flow-based techniques use this signature as elementary unit for storing traffic information: the network administrator is no longer able to see complete packets, but this might not be necessary for most applications. However, this approach cannot be used in case the payload should be available for inspection, e.g. in case of applications that detect network attacks based on some data in the packet payload (e.g. a malformed URL).

SFlow Technology:

The SFlow technology is a mixture between packet sampling and flow extraction. Packet sampling is deployed to achieve scalability and either sampled packets or the flow information related to it can be exported. This approach makes this technology suitable for a large set of environments because it allows both packet analysis (although limited to the first few hundred bytes of the packet) and flow analysis. The most important limitation is the lack of support from some of the key equipement vendors.

A common problem among the presented technologies is the impossibility to customize the set of fields being stored for each flow, which is one of the key advantages of the approach presented in this paper. Other solutions, such as standards like RMON (the IETF's remote network monitoring system) or applications like ntop, provide another way to measure network traffic. However, while they allow a network manager to determine traffic levels in network segments, total traffic loads to/from busy hosts, etc., they do not provide any flow measurement capability.

DATA COLLECTION AND STORAGE:

Saving to disk each captured packet or possibly just a snapshot of it may be feasible in some cases, but it anyway requires a significant amount of resources. Therefore, such approach cannot be considered as the basis of generic 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 approach is based on flow-based processing: 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. As mentioned earlier, a flow is a set of packets that have the same values in a given set of fields; in our approach they are not necessarily IP source/destination address, source/destination port, and protocol type — widely used as transport flow identifiers in TCP/IP networks. Our flow definition mechanism is more 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 the 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, 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. Our approach also supports netmasks (e.g. network 130.192.0.0/16). The problem of using such a coarse flow definition is that there is no way to disaggregate data. For example, the amount of traffic sent by each host cannot be known. The most relevant novelty of our approach is that the fields that are saved for each flow are completely customizable. For instance, the fields that are extracted by default in the current prototypal implementation of our solution are listed in Table 1; however, any field present in any protocol header can be extracted. The flow identification process does not require all these fields to be present at the same time: for example, ARP related fields are not present when analyzing IPv6 flows.

Protocol	Field name (s)
Ethernet	Source and destination address
Ethernet	Protocol type
VLAN	Priority
VLAN	VLAN ID
ARP	Source and destination IP address
IP / IPv6	Traffic class
IP / IPv6	Protocol type / Next header
IP / IPv6	Source and destination address
ICMP / ICMPv6	Type
TCP /UDP	Source and destination port

Table 1. Default list of fields extracted for each flow.

For each packet the probe determines the flow the packet belongs to and updates a set of counters (e.g. number of bytes/packets, timestamps, etc.). The selected fields are extracted for each flow and periodically dumped to disk together with the value of the above counters.

In order to support a variable number of fields within each flow, data is organized in three tables:

1. A transaction table keeps invariant information related to each flow;
2. An element table holds, for each flow, the list of fields to be stored;
3. The Field Type table lists all valid fields.

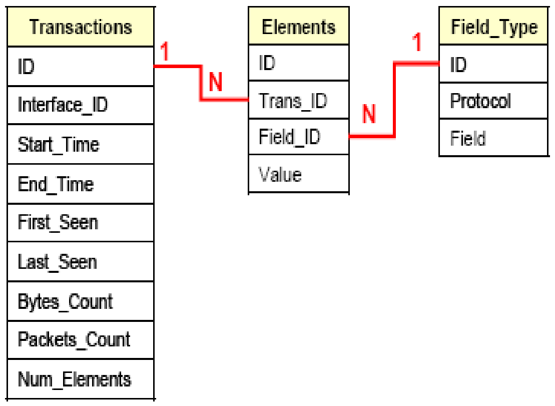
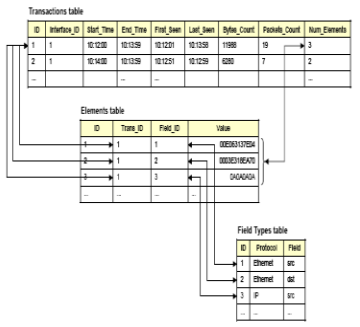


Figure 2. Structure of the database that stores network flows.

Figure 3 shows a sample table. Although this structure is slightly more complex than the traditional one (one table with a fixed number of fields, and one record per flow), it has proved much more flexible.



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 3. Snapshot of records stored in the database

SQLite was selected as a database engine since, as shown by Table 2, it provides very fast access and its overhead is only 5 times the time required to store data on a flat file (see Table 3). Since this engine cannot be configured as a standalone database server, scalability might become a problem because of the impossibility to split the load between the network probe and the database server. In order to optimize record insertion time, the flow export process dumps data on disk in a flat file and subsequently data are imported in the database through a bulk insertion. This also offers the flexibility to use a different database engine if needed, at the expense of disk space and speed.

Database Type Record/sec written on disk

Database Type	Record/sec written on disk
Access 2002	344
MySQL 3.23.53	1023
SQLite 2.8.0	11565
Text File	69657

Table 2. Number of records per seconds written on disk. Obviously, as shown in Table 3, increasing the duration of the flushing interval, further increases (even though mildly) the disk-saving factor. However, according to the figures shown in Table 3, a flushing interval larger than two minutes is not advisable since the database reduction is not significant while a longer sampling period makes computing traffic statistics harder.

	Flushing Interval		
Database Type	2 min	3 min	4 min
Access 2002	17:1	18:1	19:1
SQLite 2.8.0	19:1	20:1	20:1
SQLite 2.8.0 with indexes	10:1	10:1	11:1
Text file	29:1	31:1	32:1

Table 3. Disk space saving with different archive formats and sampling intervals.

IV.MINING 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 is represented by a record. An Item set is a set of elements – (record field, value) pairs in the database – characterized by a given value in one or more fields (e.g., IP source address and TCP source port). An Item set is considered Frequent if its cardinality exceeds a given threshold with respect to the total number of samples.For example, the set {host_dest=X, port_dest=Y} is a Frequent Itemset if there are more than Z% (e.g. 5%) samples in the database containing the set.

Association Rules are extracted from frequent itemsets and show correlations among (contained) itemsets. For instance, if a host S is active mostly as a web server, the association rule: IP dest_address = S → TCP dest_port = 80shows that there is a high probability that the flows destined to the server (characterized by the value S in the IP destination address field) contain 80 (the default TCP port for a webserver) in the TCP destination port field.

Data mining techniques are not widely used for network operation and management. Most research is related to intrusion detection systems (IDS). Lee, in [14] proposes IDS built combining various data mining techniques, thus reducing the need to manually analyze and encode intrusion patterns.

A method has been proposed to build an IDS based on clustering and anomaly detection. This method aims at dividing network traffic into clusters and then separate clusters containing normal traffic from clusters that represents intrusions, without requiring a “normal data set” to train the system. However, the assumptions on which the method is based are not realistic, thus making it of limited use in practice. The most important problem of IDS based on datamining techniques is the false positive rate, which may well be around 1%. For instance, a false positive rate of 1% with the assumption of 1 intrusion every 10,000 normal transactions, results in a false alarm ratio above 99%, which makes these methods unusable without additional techniques for false alarm reduction.

We believe that data mining techniques can be deployed much more effectively in other fields where the false positives are not an issue. In the work presented in this paper the output of the data mining process is used to create a snapshot of the network: which hosts act as servers, which ones are clients, which ones are routers, and so on. The network administrator can use the snapshot produced by the NetMiner module to check if hosts behave as expected; in addition, NetMiner can highlight changes in the network by comparing in snapshots taken at different times.

CONCLUSION:

While the data collection and storage approach can be considered stable, the data mining approach still need a more detailed evaluation and field trial. Also it is important for network administrators to be able to locate and monitor the traffic generated by these applications that are usually installed and controlled directly by network users.

On the down side, our experience with the approach shows that the interpretation of results of the data mining process is far from being straightforward. 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 hundreds 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. Thus, our work on the NetMiner module has focused on providing a user interface that, being designed specifically for network analysis applications, facilitates the network administrator in browsing through the results provided by the data mining process. More work is being done to improve this aspect of the tool. More investigation and new results are expected on an important byproduct of the proposed approach: using the outcome of the data mining process as an extremely compact representation of captured network traffic.

REFERENCES:

[1] L. Leita˜o, P. Calado, and M. Weis, “Structure-Based Inference of XML Similarity for Fuzzy Duplicate Detection,” Proc. 16th ACM Int’l Conf. Information and Knowledge Management, pp. 293-302,2007.

[2] A.M. Kade and C.A. Heuser, “Matching XML Documents in Highly Dynamic Applications,” Proc. ACM Symp. Document Eng. (DocEng), pp. 191-198, 2008.

[3] D. Milano, M. Scannapieco, and T. Catarci, “Structure Aware XML Object Identification,” Proc. VLDB Workshop Clean Databases (CleanDB), 2006.

[4] P. Calado, M. Herschel, and L. Leita˜o, “An Overview of XML Duplicate Detection Algorithms,” Soft Computing in XML Data
Management, Studies in Fuzziness and Soft Computing, vol. 255, pp. 193-224, 2010.

[5] S. Puhlmann, M. Weis, and F. Naumann, “XML Duplicate Detection Using Sorted Neighborhoods,” Proc. Conf. Extending Database Technology (EDBT), pp. 773-791, 2006.

[6] S. Guha, H.V. Jagadish, N. Koudas, D. Srivastava, and T. Yu, “Approximate XML Joins,” Proc. ACM SIGMOD Conf. Management of Data, 2002.

[7] J.C.P. Carvalho and A.S. da Silva, “Finding Similar Identities among Objects from Multiple Web Sources,” Proc. CIKM Workshop Web Information and Data Management (WIDM), pp. 90-93, 2003.

[8] R.A. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval.Addison-Wesley Longman Publishing Co., Inc., 1999.

[9] M.A. Herna´ndez and S.J. Stolfo, “The Merge/Purge Problem for Large Databases,” Proc. ACM SIGMOD Conf. Management of Data, pp. 127-138, 1995.

[10] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks ofPlausible Inference, second ed. Morgan Kaufmann Publishers, 1988.