Reducing Time-Related Demands of Flow-Based Network Monitoring Tools Emerging during Data Processing

Adrián Pekár and Martin Chovanec
Institute of Computer Technology of the Technical University of Košice, B. Němcovej 3, 042 00 Košice, Slovakia
Email: {adrian.pekar, martin.chovanec}@tuke.sk

***Abstract*—This paper deals with the time-related demands of flow-based network monitoring tools emerging during data analysis and visualization. Nowadays, most of the monitoring tools are based on flow measurement. Despite its popularity it is still surrounded by several issues, especially in case of data analysis and visualization. As networks are continuously growing in size, connected users and the volume of transmitted traffic, monitoring tools generate more and more measurement data. In consequence, processing the queries by the storage systems and the subsequent visualization of the results by the analyzing applications represent an excessive response time of the flow-based monitoring tools, thus their operation and management are becoming complex. In this paper we provide a solution for mitigating the time-related demands of flow-based monitoring tools emerging during data analysis and visualization.**

***Index Terms*—data analysis and visualization, IPFIX, MongoDB, network traffic monitoring**

I. Introduction

Nowadays, the most commonly used data measurement methods are based on collecting information about the network and its traffic at the level of flows. Network monitoring by flow-level based measurement platforms – either implementing the *NetFlow v9* [2] or *IPFIX* [3] protocols – is based on the analysis of information obtained from traffic properties and characteristics. These properties (e.g. the total number of bytes of all packets belonging to a certain flow) and characteristics (e.g. source IP address) of the flow are carried in flow records [2], [3]. The export of flow records represents a push-based mechanism, where the data are transmitted from the exporter(s) to the collector(s) over either the TCP, UDP or the SCTP protocols [2], [3]. Further important tasks, as described above, are data processing, analysis, evaluation and visualization, which most commonly take place in an *analyzing application* (analyzer).

There are many motivational factors for measuring and analyzing network traffic. Among others, flow records have a wide range of use from analyzing the traffic of the network through anomaly detection [5] up to ensuring Quality of Service (QoS). However, despite its popularity, flow-level measurement is still surrounded by several issues, especially when it comes to data analysis and visualization. As networks are continuously growing in connected users, size and the volume of transmitted data (network traffic), their management and operations are becoming more and more complex. In consequence, current flow-based network monitoring systems generate a *huge volume of measurement data* what represents one of the most critical issues from the view of both, data analysis (processing) and visualization (interpretation).

In the following Sections we provide a brief explanation of the problem we intended to contribute to; following with the description of our proposed solution; up to the summary of preliminary experiments. The last Section draws a conclusion and some future directions. Since we expect IPFIX to be the industry standard for flow monitoring in the near future, and considering the fact that between IPFIX and NetFlow is just a slight difference, in the following we will describe our approach in the context of the IPFIX specification.

II. Description of the Problem Arising during Data Analysis and Visualization

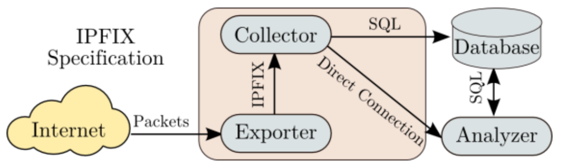


Figure 1. General architecture of a monitoring platform implementing IPFIX

Monitoring and analyzing the network traffic based on the IPFIX protocol [3], as depicted in Fig. 1, can be split into the following steps:

1) The information obtained from the captured packets after timestamping, sampling, classification, etc.; are encapsulated into IPFIX messages and sent from the exporter(s) [11] to the collector(s) [11].
2) In the collector, after parsing the currently obtained template/data record, the obtained flow-level data are stored in the database of the metering platform and/or sent directly to the analyzer.

Manuscript received August 15, 2015; revised May 3, 2016.

3) The analysis over the flow-level information is performed by an analyzing application. For example, the data obtained from a database can be processed and visualized in a form of plots. Obviously, these plots will vary according to the desired type of analysis.

However, when too many flows are present in the traffic, IPFIX-based measurement platforms have to deal with several issues. Except the lower components of the monitoring platform [13], the immense volume of measurement data (flow records) also have a high impact on the analyzing (evaluating) process(es). The time necessary for an analyzing application to get the traffic information using a database depends on many factors that are difficult to estimate. These factors, among, others are:

- The time required by the exporter to prepare and transmit the flow record to the collector.
- The time required by the collector to parse the flow record and transmit the data to the database.
- The time required by the analyzing application to process the received data from the database.

Another significant time period is the one that is consumed by the database server to store the data and return the result to the analyzer's query. This time period is highly dependent on various database technologies and data storing techniques. Moreover, with the growth of data in the database, processing the queries is becoming more and more time consuming. Unfortunately, the demands related to the storage, processing, analysis, evaluation and visualization of the flow records proportionally grow with their number.

Since the collector stores each data about the IP flows measured by the exporter(s), the size of the data in the database can be really large. For example, monitoring the network with balanced traffic during 10 days results in approximately 1.5 million of stored flow records in the database. If a user want to see what happened on the network during those days, the database system would return 1.5 million records. On the basis of these records the analyzer would have to generate a plot of the flow rate in a form of interconnected points. In the case of such a large amount of number, this generation is becoming software and hardware challenging. As a result, both, processing the queries by the database and the subsequent visualization of the results by the analyzer represent an excessive response time of the flow-based measurement tool.

In summary, one of the most critical part of flow-based monitoring architectures is their database. This is also reflected in the paper [13] describing the problems arising during network traffic monitoring. For these reasons in our work we intended to improve the way of storing the data of network traffic flows in the database and accessing/querying them.

III. Related Work

During the last decade many efforts have been placed to address the problems emerging during storing and accessing a large set of data. To improve the performance of the flow collecting mechanisms, various data compressing methods have been applied. Currently the most commonly used methods in persistent storages (ie. slower storage solution, but the most appropriate way to store data for a longer time [7]) are (i) *row-oriented databases* (or SQL) such as PostgreSQL, MySQL, etc.; (ii) *column-oriented databases* (or NoSQL) such as FastBit [6], BigTable [1], MongoDB [14] etc.; and (iii) *flat files* [4]. Each of them has advantages and disadvantages. While column-oriented databases (DBes) perform best in case of query related tasks (e.g. read), the required disk space and their overall write performance is not very impressive. Row-oriented DBes performs in comparison with the other 2 storage formats the worst. Their only advantage is the well-known and flexible query syntax. From the described 3 formats, flat files provide the best performance, however, since their speed is highly influenced by the data type (e.g. binary or text) they have to work with, this result is relative. The performance of these storage formats in case of flow collection related tasks have been compared in several works. The performance comparison of flat files and row-oriented DBes is provided in [8]. The paper by Velan [16] compared the performance between column-oriented DBes and flat files. The performance of column- and row-oriented DBes is provided in [4].

The difference between SQL (row-oriented) and NoSQL (a sub-class of column-oriented DBes) databases is in how they store the data. While SQL DBes use a fixed structure, NoSQL DBes do not have such a structure (the structure is created "on-the-fly" during writing). Since these databases differ from their basic, the manipulation with the data (more precisely the query syntax) also alters; in favor of the SQL DBes. If it comes to their performance – as showed in papers focusing on SQL and NoSQL DB performance [9], [10] – NoSQL databases are the absolute choice. The abovementioned conclusions can be driven from Table I.

TABLE I. Comparison of SQL and NoSQL Databases

Property	SQL	NoSQL
Data model	Rigidly defined	Free
Data manipulation	Standard SQL	Via API
Reliability	Native ACID	Necessary implementation
Read and write	Slow	Fast
Portability	Simpler	Complex

A promising way how to deal with large data sets in SQL databases is to deploy aggregation and summarization methods. Although this solution – as described in one of our previous works [12] – can bring positive results, in a long term – as described in our further work [13] – they still do not represent an appropriate workaround.

In conclusion, although the SQL databases might have some advantages such as the well-known SQL syntax, most of the developers got used to with, in case of such large amount of data with which one has to count in the case of network traffic monitoring, NoSQL database is an appropriate choice. Thus, in our solution we picked

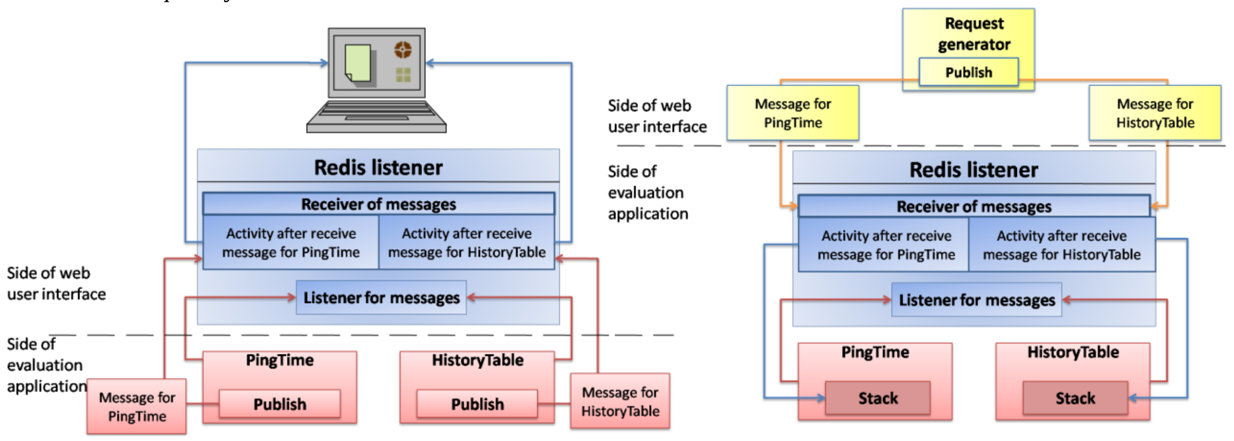
NoSQL database – specifically the MongoDB [14]. Another advantage of NoSQL databases is, that they treat records as objects, thus simplifying the implementation of the communication channel between the software components of the monitoring tool and the database.

IV. Design of the Proposed Solution

The evaluating application consists of an interface and several evaluating modules. The interface of the evaluating application is implemented by connectors which are used to connect the application to the database. The actual version of the evaluating application uses the MongoDB. As a connector between the evaluating application and the web interface the Redis database service [15] was used. To improve the speed and performance, a listener was also implemented which is listening on a pre-defined port to receive only messages designated for the concrete evaluating modules. The last component of evaluating application is a configuration file which is used to set the desired settings for the evaluating service (e.g. the information to connect to MongoDB and Redis). In addition, it also contains flags which indicate whether the module is used. As a basis an abstract module was implemented. This provides a starting point when implementing the individual modules adjusted to the specific purpose the module is performing. Obviously, since some data sets are used repeatedly, in the following we describe 3 modules, whose algorithmic complexity differs.

1) One of these modules is the one which returns the number of flows (*NumberOfFlows*). This module receives a request from the evaluating application with a filtrating criteria. On the basis of this filtrating criteria, a database query is generated and subsequently sent to the DB. From the data returned by the DB a sum from the flow identifiers is created, resulting in the number of flows.
2) Another module, with different complexity of algorithm is the one returning the amount of data transferred in the traffic (*AmountOfTransferredData*). Similarly, this module first receives a message with the filtrating criteria and creates a database query. From the returned records the sum of all the data is calculated. The computed value is sent back to the web interface via the Redis database service.
3) The last module we focused on in our work is returning the number of top uploaders (*TopUploader*). The principles of evaluating this module is similar to the previous ones, however, a further iteration is realized to get only unique IP addresses.

The main part of communication channels between the evaluating application and the web interface is the Redis database service. The user visits the user interface's page and selects a tab. Each tab belongs to a selected module. After the tab is selected the web application generates a request to all the modules which are classified in the tab. Each request is formed with a *module name* and a *filtrating criteria*. All these requests are stored in the Redis DB service. The application stores all the requests in a queue. After the module receives the request, it is processed and the result is stored in the Redis database service again. The web application subsequently pulls out the result from the query and interprets them in the GUI in a form of various plots. The mechanism of sending a request from the web interface is described in Fig. 2a. The mechanism of receiving a request from the web interface by the evaluating application is showed in Fig. 2b.


Redis listener
Receiver of messages
Activity after receive message for PingTime
Activity after receive message for HistoryTable
Listener for messages
Side of web user interface
Side of evaluation application
PingTime
HistoryTable
Publish
Publish
Message for PingTime
Message for HistoryTable
Request generator
Publish
Side of web user interface
Message for PingTime
Message for HistoryTable
Side of evaluation application
Redis listener
Receiver of messages
Activity after receive message for PingTime
Activity after receive message for HistoryTable
Listener for messages
PingTime
HistoryTable
Stack
Stack

a. Communication between the web app. and the evaluator b. Communication between the evaluator and the web app.

Figure 2. Communication channels between the web application and the evaluator

V. Preliminary Experiments

The abovementioned solution was implemented in the SLAmeter network traffic metering and evaluating tool [11]. For creating various network monitoring scenarios we realized 3 measurements, during which we collected 3 different data sets:

- One database with only approx. 500 records (noted in Fig. 3 – 5 with number 1);
- Another DB with approx. 5 000 records (noted in Fig. 3 – 5 with number 2);
- And a third one with approx. 50 000 records (noted in Fig. 3 – 5 with number 3).

Subsequently we compared the times required by the 3 modules of the tool. These modules were, as described above, the NumberOfFlows, AmountOfTransferredData and TopUploader.

The first examined module was the one generating the number of the flows – NumberOfFlows. In Fig. 3 we can see, that in the case of the smallest database having only approx. 500 records the time required for the execution of the module (aggregation on the basis of flow ID) took 37 milliseconds (ms). However, with the increase of data in the database, the time required for the evaluation/execution of the module also increased. The comparison of the smallest and the largest data sets yielded to the result that the time difference was approx. 419ms, however, the number of records was hundredfold.

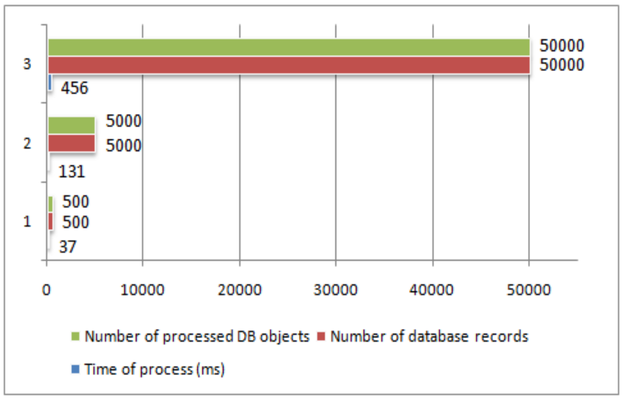


Figure 3. Chart of time dependency and the amount of data of the NumberOfFlows module

The second experiment was performed with the module determining the number of transferred data – AmountOfTransferredData. Since all the data in the database met the requirements of this module, during the execution of the module all the data in the database had to be processed. As showed in Fig. 4, the total time required for the execution of the module over the largest database was almost 4-time larger as the time in case of the second largest data set. However, if we take into account that the difference in the records is approx. 45 000, the evaluation in the case of large data set realized during almost a second is suitable.

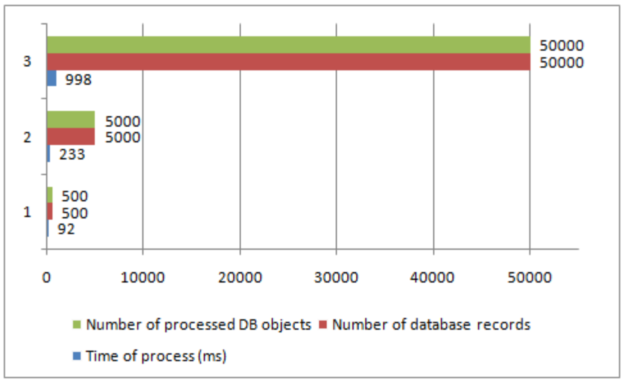


Figure 4. Chart of time dependency and the amount of data of the AmountOfTransferredData module

The last module is the module returning the number of top uploaders – TopUploaders. As shown in Fig. 5, processing a large amount of data (i.e. the DB with approx. 50 000 records) took 7 seconds. However, the algorithm for this module is more complex than in the case of the previous modules.

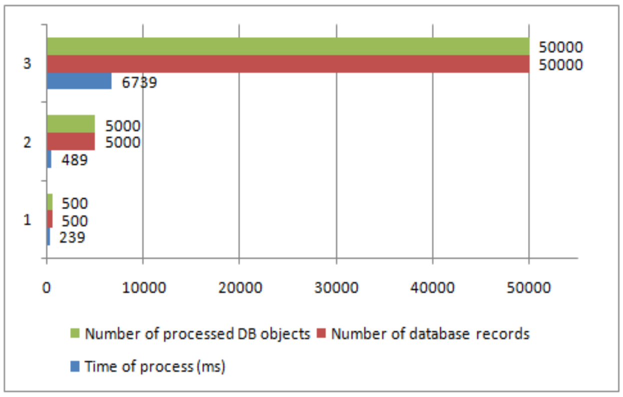


Figure 5. Chart of time dependency and the amount of data of the TopUploader module

VI. Conclusion

The constant performance increase of the monitoring systems is not a good strategy in dealing with the amount of transmitted data. Even if a small portion of the traffic is measured, the network will still have a lot more devices than the monitoring system. This results in an incomparable difference between their computation resources, i.e. the resources for traffic generation and traffic measurement. As networks are continuously growing, monitoring tools generate more and more measurement data. In consequence, processing the queries by the storage systems and the subsequent visualization of the results by the analyzing applications represent an excessive response time of the flow-based monitoring tools. In overall, this makes the operation and management of networks more complex. Therefore we can consider the database management system as one of the most critical points of the monitoring tool.

This paper was devoted to the time-related demands of flow-based network monitoring tools emerging during data analysis and visualization. We provided a solution for mitigating the time-related demands of flow-based monitoring tools emerging during data analysis and visualization. Currently there is no method by which we could measure the traffic on a per-flow basis. We provided several storage systems. Although the classical SQL databases might have some advantages such as the well-known SQL syntax, in case of a large amount of data with which we have to count in case of network traffic monitoring, NoSQL database is an appropriate choice. Thus, in our solution we picked the NoSQL database – specifically MongoDB and implemented it according to the design presented in this work. Preliminary experiments returned positive results. They showed that by appropriate design and implementation, the processing time can be reduced. In addition, due the caching mechanism of the database system, even if a module required more time at first processing, the upcoming evaluations of the same data set required significantly less.

Future work will be aimed at the realization of experiments with SQL databases in similar network conditions and their confrontation with the achieved results presented in this paper, as well as at such methods, by which the behavior of the monitoring tool can be automatically adjusted to the network state.

ACKNOWLEDGMENT

This publication is the result of the Project implementation: University Science Park TECHNICOM for Innovation Applications Supported by Knowledge Technology, ITMS: 26220220182, supported by the Research & Development Operational Programme funded by the ERDF. We support research activities in Slovakia. This project is being co-financed by the European Union.

REFERENCES

[1] B. Claise. (2004). Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational). [Online]. Available: http://www.ietf.org/rfc/rfc3954.txt

[2] B. Claise and B. Trammell. (2013). Information Model for IP Flow Information Export (IPFIX). RFC7012 (Proposed Standard). [Online]. Available: http://www.ietf.org/rfc/rfc7012.txt

[3] M. Ennert, E. Chovancová, and Z. Dudláková, "Testing of IDS model using several intrusion detection tools," *Journal of Applied Mathematics and Computational Mech.*, vol. 14, no. 1, pp. 55-62, 2015.

[4] A. Pekár, *et al.*, "Slameter – The evaluator of network traffic parameters," in *Proc. 10th IEEE International Conference on Emerging eLearning Technologies and Applications*, 2012, pp. 291-295.

[5] A. Pekár, *et al.*, "Issues in the passive approach of network traffic monitoring," in *Proc. 17th IEEE International Conference on Intelligent Engineering Systems*, 2013, pp. 327-332.

[6] R. Hofstede, *et al.*, "Flow monitoring explained: From packet capture to data analysis with Netflow and IPFIX," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2037-2064, 2014.

[7] FastBit: An efficient compressed bitmap index technology. [Online]. Available: https://sdm.lbl.gov/fastbit/

[8] F. Chang, *et al.*, "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 205-218, 2008.

[9] E. Plugge, T. Hawkins, and P. Membrey, *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing*, 1st ed., Berkely, CA: Apress, 2010.

[10] L. Deri, V. Lorenzetti, and S. Mortimer, "Collection and exploration of large data monitoring sets using bitmap databases," *Lecture Notes in Computer Science*, vol. 6003, pp. 73-86, 2010.

[11] R. Hofstede, A. Sperotto, T. Fioreze, and A. Pras, "The network data handling war: MySQL vs NfDump," *Lecture Notes in Comp. Science*, vol. 6164, pp. 167-176, 2010.

[12] P. Velan, "Practical experience with IPFIX flow collectors," in *Proc. 13th IFIP/IEEE International Symposium on Integrated Network Management*, 2013, pp. 1015-1020, 2013.

[13] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," in *Proc. 6th International Conference on Pervasive Computing and Applications*, 2011, pp. 363-366.

[14] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 2013.

[15] L. Vokorokos, *et al.*, "Preparing databases for network traffic monitoring," in *Proc. 10th IEEE International Symposium on Applied Machine Intelligence and Informatics*, 2012, pp. 13-18.

[16] R. M. Lerner, "At the forge: Redis," *Linux Journal*, vol. 2010, p. 197, 2010.

Adrián Pekár graduated at the Dept. of Computers and Informatics of the Faculty of Electrical Engineering and Informatics of the Technical University of Košice, Slovakia in 2011. Since then, his scientific research is focusing on the optimization of measurement platforms based on the IPFIX protocol. He defended his PhD thesis in the field of network traffic characteristics' measurements and monitoring in 2014. His area of interest includes QoS; IPFIX; traffic management and engineering; cloud computing and virtualization.

Martin Chovanec received his engineering degree in the field of Computer Science in 2005 at the Faculty of Electrical Engineering and Informatics (FEEI) of the Technical University of Košice (TUKE). In 2008 he received his PhD degree at the Department of Computers and Informatics of the FEEI of the TUKE. Since then his scientific research is focused on network security and encryption algorithms. Currently he is the director of the Inst. of Computer Technology of the TUKE.