Practical Experience with IPFIX Flow Collectors

Petr Velan
CESNET, z.s.p.o.
Zikova 4, 160 00 Praha 6, Czech Republic
petr.velan@cesnet.cz

***Abstract*—As the number of Internet applications grows, the number of applications that use data encapsulation increases as well. Flow monitoring using NetFlow version 5 or 9 is only able to analyze the encapsulating protocol, therefore it becomes too limited to detect new threats. For this reason, more thorough knowledge of such traffic is needed. The IPFIX protocol can be used in such situations, because it provides enough flexibility for monitoring tools to be extended by new elements. Along with greater flexibility, IPFIX support results in a higher performance footprint on collectors and tools for querying the collected data. Currently, there is a lack of flow collection frameworks with IPFIX support that would allow flow data to be extended. The aim of this paper is to compare open-source flow collectors that provide support for the IPFIX protocol. We focus on evaluating performance of query tools and the level of IPFIX support provided by the collection frameworks.**

I. Introduction

The amount and complexity of network traffic are constantly growing and network monitoring is therefore crucial for management of large networks. Flow monitoring is often used for this purpose, because it is able to handle large volumes of traffic, which is possible due to its distributed architecture. In the most common configuration, flow records containing aggregated information about whole connections are created by flow exporters and sent to a collector using the NetFlow [1] protocol. NetFlow v9 [2] is currently the most popular version of the protocol, because it defines a large set of field types that can be used for flow description. However, there is no support for any extensions and thus no new field types could be created as needed. This problem is solved in the IPFIX [3] protocol, which is a next generation of NetFlow. IPFIX allows definition of enterprise elements that can be used to enrich the flow records with new information.

In the IPFIX protocol, field types are known as Information Elements. Further in the paper, we will refer to them as elements for the sake of brevity. IPFIX extends NetFlow v9 by adding new elements while preserving the old ones to maintain compatibility. Moreover, IPFIX uses Private Enterprise Numbers to define enterprise elements. The Private Enterprise Numbers are assigned to organizations by IANA [4], therefore each organization can define its own enterprise elements without a risk of collisions.

The IPFIX enterprise elements are used to gain more information about individual flows. This provides better understanding of the traffic in the network, for example by adding DNS payload to DNS queries or by adding geolocation to the flows. The support of IPFIX enterprise elements is therefore crucial for a future development of advanced flow-based network monitoring systems. Unfortunately, IPFIX support provided by the currently used collection frameworks is rather limited and is not sufficient for this purpose.

Higher flexibility of IPFIX protocol often affects data storage format of IPFIX collection frameworks, since the collectors must be able to store newly defined elements transparently. This can have non-negligible impact on performance of data processing tools and should be taken into account when choosing IPFIX collection framework.

The goal of the paper is to present a comparison of the most widely used flow collection frameworks with IPFIX support. The comparison should provide information to help with choosing the right framework to suit specific needs, which can arise when flow collection solution requires flexibility of the IPFIX protocol. We are looking for collection frameworks that would enable us to add new elements to flow records, store them and use them in future data mining. We limit the selection of the frameworks to those that are provided as open-source, so that we can use them freely. In particular, we answer the following questions:

1) Which open-source IPFIX collection frameworks are available?
2) What is the level of IPFIX support that is provided by each collection framework?
3) What is the performance of query tools that come with the collection frameworks?
4) To what extent are the query tools in collection frameworks extensible?

To answer these questions, we need to find out which common collection frameworks provide IPFIX support. We will focus on non-commercial collectors with a freely available source code. Some of the chosen frameworks were presented at the IPFIX Interoperability Event [5] that took place in May 2011, others are known to us from our own experience. After that, we will study documentation of the collectors to determine the level of the support and we will verify the results in practice. For that purpose, we will deploy the evaluated collectors on a dedicated machine and use them to store IPFIX flows. Then we will use the stored data to measure the performance of query tools provided by each of the collection framework.

The rest of the paper is structured as follows. In Section II we survey the most common collection frameworks with IPFIX support and select collectors relevant to our research. Section III discusses the extent of IPFIX support provided by chosen collectors. The performance of the query tools is described in Section IV. Section V is dedicated to comparison of extensibility of query tools in collection frameworks. The paper is summarized in Section VI and conclusions are drawn.

978-3-901882-50-0

II. IPFIX Collection Frameworks

There are several collection frameworks that provide some level of IPFIX support. One of them is *ntop* [6], which is a tool for traffic analysis with support for NetFlow a IPFIX. The primary purpose of *ntop* is to show local network usage. Therefore it works as a combination of an exporter and a collector, where the exporter is processing data on local machine. However, it is possible to use the collector and exporter features separately to collector flow data from other exporters or to export flows to another collector. The tool provides Web interface, which enables sorting and aggregating network data by various criteria and can be also used for configuration and administration of the *ntop*. The Web interface also shows network data statistics, which are persistently stored in the RRD format, which is the only persistent storage that *ntop* provides.

From the same authors as *ntop* comes *nProbe* [7], which is an extensible NetFlow exporter and collector. It was designed to be deployed in embedded environments, so it has no Web interface. Exporter mode, collector mode and proxy mode are available in *nProbe*. The proxy mode is designed to receive and resend the flow data in another version of NetFlow format, including IPFIX. Every mode allows storage of the processed flow data to disk in MySQL, text, binary or FastBit [8] format. *nProbe* provides full support for IPFIX Private Enterprise Numbers and variable length encoding.

Similar architecture to *nProbe* provides *Vermont* [9]. Its primary purpose is to be a flow exporter, but it also provides features for extensive flow manipulation and it can also be used as a proxy for receiving, processing and resending flow data in NetFlow and IPFIX formats. The flow processing is done using different plugins, which are able to do sampling, anonymization and other tasks. There are also plugins that allow storage of the processed flows to MySQL and Oracle databases.

The rest of the collection frameworks do not include flow exporters. The *SiLK* [10] framework is a collection of more than fifty tools that are designed to capture flows in NetFlow or IPFIX format and store them for later usage. *SiLK* uses its own format to store the flow records which also supports data compression. This approach allows it to be more distributed, because it has one tool to process incoming flows and resend the transformed data to another tool, which handles the actual storage. Basic data classification is done during the storage, so that the flows are sorted into different classes based on various properties. The data in the classes can be queried by multiple tools. Each is dedicated to a specific task, so there are tools for filtering stored data, for grouping the results and for formatting the output. Together the whole framework provides a complete solution for flow data management.

Another collection framework is called *nfdump* [11]. Unlike *SiLK*, *nfdump* provides one tool for a flow data collection and storage and one tool that can issue queries on the stored data. In addition, there are several other tools that can be used for resending the flows, anonymization, or management of old flow data files. The format used to store flow data is specific to *nfdump* and data compression is supported. The *nfdump* has support for NetFlow and IPFIX protocols.

Each of the mentioned frameworks was designed to work with NetFlow v9 and IPFIX support was added later. A different approach is used by *IPFIXcol* [12], which is a collection framework that was developed specifically for the IPFIX protocol. The *IPFIXcol* is a framework that uses input and output plugins to acquire desired functionality. Input plugins receive data using different protocols and pass them to the *IPFIXcol* core. The core performs basic data processing, handles templates and passes the data to output plugins. The output plugins can resend or store the data. This design is flexible and allows the same data to be processed by multiple plugins simultaneously, which can be used for example to store the data in one plugin while creating real-time statistics in another. Currently the *IPFIXcol* framework provides a storage plugin and a query tool for FastBit database.

FastBit is a column-oriented database, therefore the same IPFIX elements are stored in a single column, represented by a single file. Different IPFIX template are stored in different directories, which allows easy addition of new elements as necessary. Another advantage of FastBit database is that it provides support for compressed bitmap indexes, which are used to speed up the data access. This results in higher performance of the query tools that use the FastBit database.

To the best of our knowledge, the presented open-source IPFIX collection frameworks are currently the only ones that are actively developed and maintained. We have deployed flow exporters on all external links that connect our network (CESNET – Czech national research and education network) with others. All of the links are high-speed (10 Gbps), so we distribute the load by separating exporters from collector. Stored flows are further processed at the collector, to provide a threat detection or a long term statistics. Collection frameworks that match our setup are *nfdump*, *SiLK* and *IPFIXcol*. We will use *IPFIXcol* with the FastBit output plugin. The *ntop* is excluded since it does not store raw flow data. The reason not to include *nProbe* and *Vermont* in our comparison is that both are flow exporters by design. There are also performance issues when using DBMS for flow data storage, as shown in [13]. Although the FastBit support provided by *nProbe* would be beneficial, because it allows storage of enhanced flow records, the FastBit support is not included in the freely available version. Moreover, the flexibility of flow data storage based on fastbit FastBit is described as a part of *IPFIXcol*. Although we will not compare the performance of *Vermont* in this work, since it would duplicate the [13], we will include it in the comparison in Section III.

III. IPFIX Support Assessment

In this section, we discuss the level of IPFIX support provided by each collection framework. Since there are many IPFIX elements already defined by IANA [14], it is important to know whether the frameworks are able to store and process them. As we described earlier, we are looking for ways of putting additional information to flow records in order to gain a greater understanding of the traffic in our network. Therefore the extensibility of the frameworks is significant and we need to study it to find out whether the frameworks support this goal. Table I lists the software used along with its version.

TABLE I. VERSIONS OF THE SOFTWARE USED

Software name	Version
nfdump	1.6.6
SiLK	2.5.0
IPFIXcol	0.5.1
FastBit	SVN revision 532

A. IPFIX Elements Support

Every framework supports basic flow information, such as source and destination IP addresses and ports, protocol, flow timestamps or duration, TCP control bits, number of bytes and packets. In addition to these, other elements are supported by each framework, the Table II shows a basic comparison of the additional element support. It is based on documentation, source code and our practical experience.

In *IPFIXcol*, the extent of the supported IPFIX elements is limited only by output plugin in use, because the plugin is provided with all of the received data. The FastBit output plugin is able to store any IPFIX element defined by IANA, with the exception of elements with octetArray, basicList, subTemplateList and subTemplateMultiList data types. The octetArray support is planned in future releases of the plugin, however, the current level of IPFIX support already allows most of the IPFIX elements to be stored, since a string type can be often used instead of octetArray. The list types are used only as generic elements and no special meaning is assigned to them yet.

The *SiLK* framework supports 29 of the IANA defined elements, which are stored as 14 *SiLK* elements. The reason for this is that sometimes different IPFIX elements are used for the same information. The elements that define flow start and flow end timestamps are a good example, since there are 5 elements for one timestamp, each using different units. In addition to the defined elements, *SiLK* is able to process some enterprise elements as well. These elements are exported by YAF [15] and are used to identify the exporter, flow type and application, or to provide more detailed information about TCP flags. YAF is a flow exporter that is developed by same authors as SiLK, so that the tools are adapted to work well with each other.

Even more elements are supported in *nfdump*. It is able to store 45 different IANA defined IPFIX elements, all of them compatible with NetFlow v9. Apart from these, there is support for 9 elements that were defined in NetFlow v9 and that are marked as reserved in IPFIX. The last element that can be stored in the *nfdump* data files is the IP address of an exporter. IPFIX support in *nfdump* should be transparent, so that it should process IPFIX elements in the same way as NetFlow, but the support is still experimental, so that for example sampling for IPFIX is not implemented yet.

The *Vermont* framework uses database systems to store flow data. Since data storage is not the primary purpose of *Vermont*, the list of supported elements is rather short. On the other hand, *Vermont* enables storage of reverse elements as well, so that the flows that represent both directions of the communication can be stored as one record. Apart from this, *Vermont* is also able to store exporter identifier and maximum packet gap. Although the number of supported elements is limited, the ability to store bidirectional flows is not present in *SiLK* nor *nfdump*.

TABLE II. SUPPORTED IPFIX ELEMENTS

Element	ndump	SiLK	Vermont	IPFIXcol
Reverse elements	no	no	yes	yes
Flow end reason	no	yes	no	yes
Vlan ID	yes	yes	no	yes
Next hop IP	yes	yes	no	yes
Forwarding status	yes	no	no	yes
SNMP	yes	no	no	yes
Autonomous sys.	yes	no	no	yes
MPLS	yes	no	no	yes
Exporter IP	yes	no	no	yes
BGP Next Hop IP	yes	no	no	yes
Mac addresses	yes	no	no	yes
Flow direction	yes	no	no	yes
Enterprise elements	no	limited	no	yes

B. Collection Framework Extensibility

While it is important for collection frameworks to have a good support of basic IPFIX elements, we are interested in the extensibility of the collectors even more. One of the real-world requirements for an IPFIX flow collector is the ability to process and store new elements on demand without any reconfiguration.

We have already shown that *IPFIXcol* has support for almost any IPFIX element, but the collector is not limited to the defined elements only. Since it uses flexible data storage, it is able to process and store even enterprise elements, as long as the definition of the elements type is provided. Once *IPFIXcol* knows the data type of a new element, it can store it accordingly using the FastBit database. The *SiLK* framework does not provide any such features, the described element set is strictly determined by the data file format. The only way of affecting the data format is to compile the *SiLK* with IPv6 support, which makes the IP fields big enough for IPv6 addresses. Although *Vermont* uses a relation database to store the data, the set of elements that can be stored is fixed as well. The *nfdump* uses a different approach. Since it provides a lot of elements, storing all of them in each flow record would be inefficient and would consume too much disk space. Instead, *nfdump* separates the additional elements to extension sets, which can be used as needed. This allows only the elements that are actually used to be stored, therefore saving disk space. Although it is not possible for users to define new extension sets, the authors are able to extend the data file format with new elements without breaking the backward compatibility.

IV. QUERY TOOL PERFORMANCE

In this section we compare the performance of query tools that are provided by each collection framework. We measure query response time of the tools on a defined set of queries. The response time is measured as time between the start of the query and a complete result set retrieval, although the results are not printed on the screen. The queries that were issued on the data set are shown in Table III. All queries in Table III were translated into the specific query language of each collection framework. Query *Q1* returns the total sum of

TABLE III. MEASUREMENT QUERIES

Q1	SELECT count(*), sum(packets), sum(bytes) FROM dataset
Q2	SELECT count(*) FROM dataset WHERE dst_port = 53
Q3	SELECT date_start, protocol, src_IPv6, dst_IPv6, src_port, dst_port, packets, bytes FROM dataset WHERE dst_port = 53 AND ip_version = 6
Q4	SELECT src_IPv4, packets, bytes, count(*) FROM dataset WHERE ip_version = 4 GROUP BY src_IPv4 ORDER BY bytes DESC LIMIT 5
Q5	SELECT protocol, src_IPv4, dst_IPv4, src_IPv6, dst_IPv6, src_port, dst_port, packets, bytes, count(*) FROM dataset GROUP BY protocol, src_IPv4, dst_IPv4, src_IPv6, dst_IPv6, src_port, dst_port

packets, bytes and flows in the data set. It measures response time of aggregation performed on all flow records using only a subset of columns. In *nfdump*, the results are already present in the header of each file, but if any filter is applied, they must be computed from the stored data. This is demonstrated in Query *Q2*, where a number of flows with destination port 53 is computed. This is an example of a query where we are only interested in the number of results. The third query lists all IPv6 flows with destination port 53, so that the performance of filtering and printing of the results is measured. The query *Q4* is used to find the top five talkers grouped by source address, which compares the performance of aggregation and ordering. And finally, the query *Q5* returns flows aggregated by standard 5-tuple, so that the aggregation performance is compared on more data. The query in SQL notation from the Table III is simplified, since otherwise we would have to perform a union of queries for IPv4 and IPv6. To show how the tools cope with increasing data sets, we ran the queries on increasingly larger subsets of the data. The subsets were selected by time intervals; first hour of traffic, first two hours, and so on, until the full data set was used.

The performance of IPFIXcol using FastBit database is measured using two different data storage settings. The first has flow records stored in the order of their arrival, the other uses FastBit reorder functionality to sort the data set by the value of the columns, starting with columns with the least value range. This process can significantly reduce the size of bitmap indexes, because the compression is more effective with ordered data. It can also increase performance of the queries, since the data locality is increased by storing flows with same element values near each other, therefore reducing the number disk read operations that are needed to retrieve the results. In our measurements, the index size was decreased to 26% of its original size. Another advantage of having reordered data is that it is also faster to build indexes on it. The comparison of FastBit performance with *nfdump* and MySQL is done in [16]. However, the data set used for measurements was significantly smaller compared to ours. We show that the FastBit library has limits that were not reached by the authors.

The flow data that are used in the measurements are taken from one of the CESNET metering points and contain 24 hours of network data measured on a common working day. The data set consists of almost 1.1 billion flow records and the size of the whole data set in *nfdump* and *IPFIXcol* format is almost 52.7 GB. However, the *SiLK* data set takes more than 67.2 GB. This is because the file format of the *SiLK* always stores the same set of elements, regardless of whether they are used or not. Therefore there are often unused elements stored in the flow record, e.g. IPv6 addresses in IPv4 flow record.

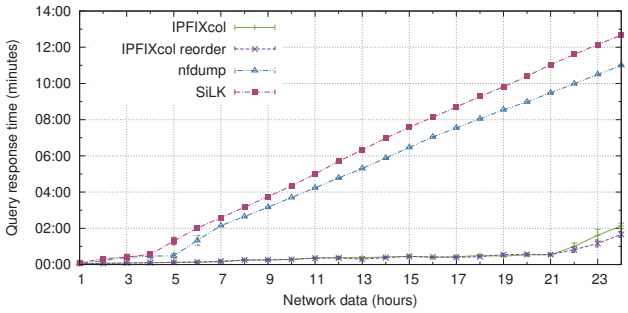

Fig. 1. Query Q1 Results

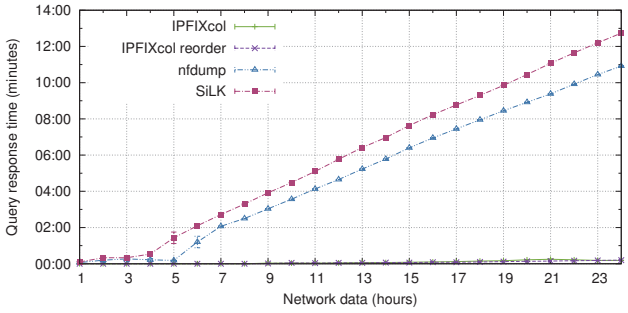

Fig. 2. Query Q2 Results

The index created by FastBit on a destination port has 6.5 GB on the original data and less than 1.7 GB on the reordered data. No compression is used in *nfdump* or *SiLK*, since there is no support for compression in the FastBit library, therefore we provide similar conditions for all tools. The compression might actually increase the performance of *nfdump* and *SiLK* queries, since the bottleneck for these tools is the disk read operation. All measurements have been performed on a machine with 2x Intel Xeon E5410 CPU working at 2.33 GHz, 12 GB DDR2 RAM and single SATA disk Seagate Barracuda ES.2 with 7200 RPM.

The results of the measurements are shown in Figures 1, 2, 3, 4 and 5, where the response time for a given interval of flow data is displayed. The queries were executed three times and the graphs show average values with corresponding error bars that represent standard error values. The *nfdump* framework has a better response time by almost two minutes for every query, as shown in Figures 1 to 5. The progress of the graphs indicates that the difference might be even greater with larger data sets. The comparison of *IPFIXcol* with and without reordered data shows that the increased performance is recognizable mainly in query *Q4*, shown in Figure 4, where the data locality is helpful when accessing only records with a specific port number.

The most notable results are gained from a comparison of *IPFIXcol* with other frameworks. *IPFIXcol* reads only columns necessary to answer a query, which results in lesser disk read operations. This can shorten the response time significantly, as shown in Figure 1. The result of query *Q2* can be found in Figure 2. The low response time of *IPFIXcol* is caused by the use of bitmap index on a destination port. Since only a number of matching flows is required, only the index needs

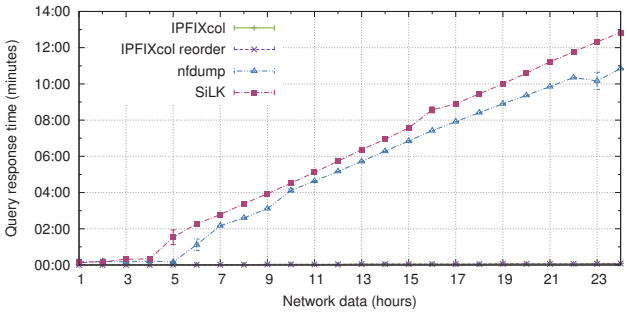

Fig. 3. Query Q3 Results

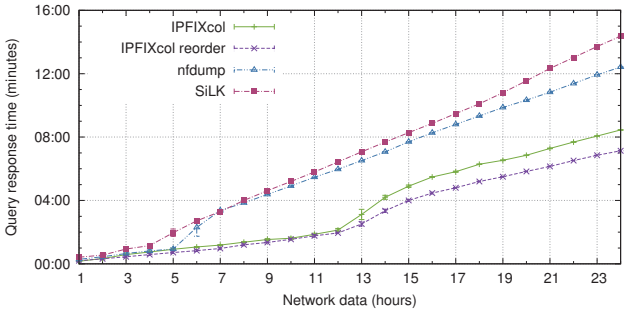

Fig. 4. Query Q4 Results

to be accessed, which is significantly faster than reading the whole data set from disk.

The third query requires that the flow data are accessed. On the other hand, the filter that selects only IPv6 flows is quite restrictive. Therefore the FastBit indexes significantly help to locate and read the necessary data only. As shown in Figure 3, this causes *IPFIXcol* to answer the query almost instantly. Moreover, since different templates are stored in different directories, the query tool might be optimised to process only directories which contain the column for filtered element. In the fourth query, flows are read and aggregated. In Figure 4 we can see that the query response time of *IPFIXcol* is increasing as fast as that of the other tools. The reason is that there is no index involved in the query other than the filtering on IPv4 protocol, which is not restrictive enough to help significantly. However, not all of the elements are accessed, which results in lower response times for *IPFIXcol*.

Figure 5 shows an example of a query, where *IPFIXcol* is not nearly as effective as the other tools. The task of aggregating all flows from a given time window requires a lot of memory and the algorithms for the aggregation of large data sets are not as optimized in FastBit library as in the *nfdump* or *SiLK*. The aggregation of four hours takes about 20 minutes and for larger data sets the FastBit library runs out of memory and ends the query execution. This is a problem of the aggregation implementation in the FastBit library. Once this is amended, this query should be answered in a time similar to that of the other tools.

The queries demonstrated that the query tools utilizing indexes for faster data access can achieve significantly lower query response time. Therefore it is vital to choose adequate

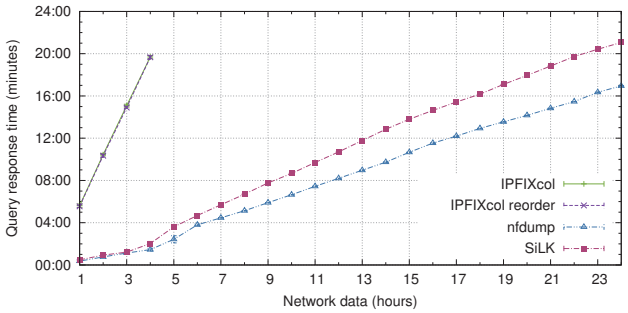

Fig. 5. Query Q5 Results

data storage for IPFIX collection framework in order to achieve the best performance.

V. Query Tool Extensibility

The ability of collection frameworks to store new elements was already discussed in Section III. This section describes the extensibility of the query tools provided by the frameworks. When support for a new IPFIX element is added to the collector, the query tools must know the semantics of the element in order to be able to work with it correctly. The element must be properly formatted on output and the user must be able to issue queries based on the elements value. For example the TCP control flags are internally stored as byte value, but the element is usually displayed and manipulated as a set of letters corresponding to each of the bits.

IPFIXcol framework provides a query tool that works with FastBit database. Since the data in FastBit database are stored using basic integer types, strings or binary arrays, the tool needs to have a definition of semantics for every element it processes. For that reason, there is an XML configuration file that allows specification of the columns that are later used in the tool. Each column is either bound to an IPFIX element or it can be a function of other columns. This approach is used to create computed columns like bytes per seconds or duration. The configuration file also allows specification of multiple aliases for each column, by which the columns are referenced on the command line of the query tool. The tool comes with a predefined set of columns that can be easily expanded, however, the list of semantics that can be assigned to new elements is fixed.

SiLK allows its tools to be extended using Python plugins. These plugins can be used to write filtering rules so that the common operations can be simplified. It is also possible to use the plugins to display new fields computed from existing data, or to extend grouping and sorting functionality. Although the plugins can be used to define semantics for new IPFIX elements, the storage file format still restricts *SiLK* from being able to add new elements on demand. Unlike *IPFIXcol*, *nfdump* does not provide any means of extension of the existing set of elements. Although its file format supports extensions, the changes to support new elements have to be made to the original source code. There is no way to define new semantics for existing elements either, as can be done in *SiLK*, since *nfdump* does not provide support for plugins.

VI. Conclusions

The paper presented an analysis of IPFIX support in current open-source collection frameworks. The IPFIX protocol allows definition of enterprise information elements, that can be used to export additional information about flows. This can be used to gain better knowledge of the network traffic. We focused on the level of IPFIX support, extensibility and query tools performance that are provided by the frameworks. We have chosen to compare *nfdump*, *SiLK* and *IPFIXcol*, because these collection frameworks provide their own tools to query collected data and are freely accessible. *Vermont* framework was also included in comparison of IPFIX element support, even though it uses DBMS to store flow records.

The level of IPFIX support provided by each collection framework varies. *Vermont* supports only a fixed set of IPFIX elements, although the relational database could be used to build a more flexible storage. The data storage used by *SiLK* is not designed to handle flexible flow data and it supports only a limited set of elements. The file format used by *nfdump* provides support for extensions, thus it supports more IPFIX elements than *SiLK*. Moreover, using the extensions to store only the elements that are actually used, the *nfdump* file format is also more disk space efficient. The *IPFIXcol* framework uses FastBit database to store flow data. Since the database is column-oriented, new elements are added as new files, so that *IPFIXcol* is able to store not only IPFIX elements defined by IANA, but also enterprise elements defined by user.

We measured the performance of query tools using a defined set of queries designed to test different aspects of the tools functionality. The results show that *nfdump* has a better query response time than *SiLK*. In the comparison of *IPFIXcol* with *nfdump* and *SiLK*, the *IPFIXcol* had a shorter query response time in most of the queries. Therefore we can deduce that when FastBit is not required to read all columns or is able to use indexes for filtering, the time needed to answer a query is significantly shorter. Only the algorithm for aggregation is not as memory efficient as that of the other tools, therefore when using multiple elements to aggregate a large set of flow data, the FastBit library may run out of memory. This is a known problem of the library and is likely to be fixed in the future.

When choosing storage for flow data, it is necessary to find a point of balance between data size, ease of access and flexibility of the storage. The FastBit database is a good example of this. The indexes speed up access to the data and the storage format is very flexible, since it is column based. On the other hand, the indexes occupy additional space and the FastBit format uses a lot of small files, which is more difficult to manage than a single file storage.

Query tools that come with *SiLK* can be extended by plugins to support new filtering clauses and new semantics for an existing set of elements. However, due to the fixed data storage format, there is no support for addition of new IPFIX elements. *IPFIXcol* allows addition of support for new elements in queries, but only existing semantics can be used for the elements. There is no support to add new IPFIX elements or semantics to existing elements in *nfdump*, unless the source code is modified.

The current level of IPFIX support provided by most of the existing collection frameworks is limited to support only some of the NetFlow v9 elements. The purpose of the IPFIX is to provide means to use elements created by users, so that the extended flow data can be used for more efficient network monitoring. Only the *IPFIXcol* allows enterprise elements to be stored and used efficiently in queries, although the lack of data compression increases storage requirements. The other collection frameworks need further development to be able to fully support the IPFIX protocol.

Acknowledgement

This material is based upon work supported by the "CESNET Large Infrastructure" project LM2010005 funded by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] Cisco Systems. Cisco IOS NetFlow. Accessed on 10 September 2012. [Online]. Available: www.cisco.com/go/netflow

[2] B. Claise, "Cisco Systems NetFlow Services Export Version 9," RFC 3954 (Informational), October 2004.

[3] B. Claise, "Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information," RFC 5101 (Proposed Standard), Internet Engineering Task Force, January 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5101.txt

[4] IANA. Accessed on 10 September 2012. [Online]. Available: http://www.iana.org/about

[5] DEMONS IPFIX Interoperability Event – Final Report (2011). Accessed on 10 September 2012. [Online]. Available: http://www.ietf.org/proceedings/80/slides/ipfix-4.pdf

[6] L. Deri. ntop. Accessed on 10 September 2012. [Online]. Available: http://www.ntop.org/products/ntop/

[7] L. Deri. nProbe. Accessed on 10 September 2012. [Online]. Available: http://www.ntop.org/products/nprobe/

[8] Lawrence Berkeley National Laboratory. FastBit. Accessed on 10 September 2012. [Online]. Available: http://crd-legacy.lbl.gov/~kewu/fastbit/

[9] FAU Erlangen and TU München. Vermont. Accessed on 10 September 2012. [Online]. Available: https://github.com/constcast/vermont/wiki

[10] CERT Network Situational Awareness Team. SiLK. Accessed on 10 September 2012. [Online]. Available: http://tools.netsa.cert.org/silk/index.html

[11] P. Haag. nfdump. Accessed on 10 September 2012. [Online]. Available: http://nfdump.sourceforge.net

[12] P. Velan and R. Krejčí, "Flow Information Storage Assessment Using IPFIXcol," in *Dependable Networks and Services*, ser. Lecture Notes in Computer Science, R. Sadre, J. Novotný, P. Celeda, M. Waldburger, and B. Stiller, Eds. Springer Berlin / Heidelberg, 2012, vol. 7279, pp. 155–158.

[13] R. Hofstede, A. Sperotto, T. Fioreze, and A. Pras, "The Network Data Handling War: MySQL vs. NfDump," in *16th EUNICE/IFIP WG 6.6 Workshop on Networked Services and Applications - Engineering, Control and Management, Trondheim, Norway*, ser. Lecture Notes in Computer Science, vol. 6164. Berlin: Springer Verlag, June 2010, pp. 167–176.

[14] IP Flow Information Export (IPFIX) Entities. Accessed on 10 September 2012. [Online]. Available: http://www.iana.org/assignments/ipfix/ipfix.xml

[15] CERT Network Situational Awareness Team. YAF. Accessed on 10 September 2012. [Online]. Available: http://tools.netsa.cert.org/yaf/index.html

[16] L. Deri, V. Lorenzetti, and S. Mortimer, "Collection and Exploration of Large Data Monitoring Sets Using Bitmap Databases," in *Traffic Monitoring and Analysis*, ser. Lecture Notes in Computer Science, F. Ricciato, M. Mellia, and E. Biersack, Eds. Springer Berlin Heidelberg, 2010, vol. 6003, pp. 73–86.