Collection and Exploration of Large Data Monitoring Sets Using Bitmap Databases

Luca Deri[1 2], Valeria Lorenzetti[1], Steve Mortimer[3]

[1] ntop.org, Italy
[2] IIT/CNR, Italy
[3] British Telecom, United Kingdom
{deri, lorenzetti}@ntop.org, steve.mortimer@bt.com

Abstract. Collecting and exploring monitoring data is becoming increasingly challenging as networks become larger and faster. Solutions based on both SQL-databases and specialized binary formats do not scale well as the amount of monitoring information increases. This paper presents a novel approach to the problem by using a bitmap database that allowed the authors to implement an efficient solution for both data collection and retrieval. The validation process on production networks has demonstrated the advantage of the proposed solution over traditional approaches. This makes it suitable for efficiently handling and interactively exploring large data monitoring sets.

Keywords: NetFlow, Flow Collection, Bitmap Databases.

1 Introduction

NetFlow [1] and sFlow [2] are the current state-of-the-art standards for building traffic monitoring applications. Both are based on the concept of traffic probe (or agent in the sFlow parlance) that analyzes network traffic and produces statistics, known as flow records, which are delivered to a central data collector [3]. As the number of records can be pretty high, probes can use sampling mechanisms in order to reduce the workload on both probe and collectors. In sFlow, the use of sampling mechanisms is native in the architecture so that it can be used by agents to effectively reduce the number of flow records delivered to collectors. This practice has a drawback in terms of result accuracy while providing them with quantifiable accuracy. In NetFlow the use of sampling (both on packets and flows) while reducing the load on routers it leads to inaccuracy [4] [5] [6], hence it is often disabled in production networks. The consequence is that network operators have to face the problem of collecting and analyzing a large number of flow records. This problem is often solved using a flow collector that stores data on a relational database or on a disk in raw format for maximum collection speed [7] [8]. Both approaches have pros and cons; in general SQL-based solutions allow users to write powerful and

expressive queries while sacrificing flow collection speed and query response time, whereas raw-based solutions are more efficient but provide limited query facilities.
The motivation behind this work is to overcome the limitations of existing solutions and create an efficient alternative to relational databases and raw files. We aim to create a new generation of a flow collection and storage architecture that exploits state-of-the-art indexing and querying technologies [9], and a set of tools capable of interactively exploring large volume of collected traffic data with minimal query response time.

The main contributions of this paper include:

- The ability to execute multidimensional queries on arbitrary large amounts of data with response time in the order of seconds (in many cases, milliseconds).
- An efficient yet simple flow record storage architecture in terms of disk space, query response time, and data collection duration.
- A system that operates on raw flow records without first reducing or summarizing them.
- The reduction of the time needed to explore a large dataset and the possibility to display query results in real-time, making the exploration process truly interactive.

The following section presents a survey of relevant flow storage and retrieval architectures, describes their limitations, and lists a set of requirements that a flow management architecture should feature. Section three covers the architecture and design choices of the proposed solution. Section four validates this solution on two production networks, evaluates the implementation performance and positions this work against popular tools identified during the survey.

2 Related Work and Motivation

Flow collectors are software applications responsible for receiving flow records emitted by network elements such as routers and switches. Their main duty is to make sure that all flow records are received and successfully written on a persistent storage. This solution limits flow record loss and decouples the collection phase from flow analysis, with the drawback of adding some latency as records are often not immediately processed as they arrive. Tools falling into this category include nfdump [10], flow-tools [11], FlowScan [12], Stager [13] and SiLK [14]. These tools store data in binary flat files, optionally in compressed format in order to reduce disk space usage and read time; they typically offer additional tools for filtering, extracting, and summarizing flow records matching specific criteria. As flat files have no indexing, data searching always requires a sequential scan of all stored records. In order to reduce the dataset to scan, these tools save flow records in directories that have a specific duration, so that to ease record temporal selection during queries. Basically the speed advantage of dumping flow records in raw format is paid at each search operation in terms of amount of data to read. Another limitation of these families of tools, is that the query language they offer is limited when compared to SQL, as they feature flow-based filtering with minimal aggregation, join and reporting facilities.

The use of relational databases is fairly popular in most commercial flow-collectors such as Cisco NetFlow collector, Fluke NetFlow Tracker, and on open-source tools such as Navarro [15] and pmacct [16]. The flexibility of the SQL language is very useful during report creation and data aggregation phases although some researchers have proposed a specialized flow query language [17]. Unfortunately the use of relational databases is known to be slower (both during data insert and query) and take more space when compared to raw flow record files [18] [19] [20].

The conclusions of the survey on popular flow management tools are:

- Tools based on raw binary files are efficient when storing flow records (e.g. nfdump can store over 250K records/sec on a dual-core PC) but provide limited flow query facilities.
- Relational databases are both slower during flow record insertion and retrieval, but thanks to SQL they offer very flexible flow query and reporting facilities.
- On large volume of collected flow records, the query time of both tool families takes a significant amount of time (measured in minutes if not hours [21]) even when high-end computers are used, making them unsuitable for interactive data exploration.

Seen that the performance figures of state-of-the-art tools is suboptimal, authors investigated whether there was a better solution to the problem of flow collection and query with respect to raw files and relational databases.

2.1 Towards Column-oriented Databases with Bitmap Indexes

A database management system typically structures data records using tables with rows and columns. The system optimizes the query-answering process by implementing auxiliary data structures known as database indexes [22] to accelerate queries. Relational databases encounter performance issues with large tables in particular because of the size of table indexes that need to be updated at each record insertion. In the last few years, new regulations that require ISPs to maintain large archive of user activities (e.g. login/logout/radius/email/wifi access logs) [23] stimulated the development of new database types able to efficiently handle billion of records. Although available since late 70‘s [24], column-oriented databases [25] have been niche products until vendors such as Sensage [26], Sybase [27] and open source implementation such as FastBit [28] [29] [30] ignited new interest on this technology. A column-oriented database stores its content by column rather than by row is known as vertical organization. This way the values for each single column are stored contiguously, and column-stores compression ratios are generally better than row-stores because consecutive entries in a column are homogeneous to each other [31] [32]. These database systems have been shown to perform more than an order of magnitude better than traditional row-oriented database systems, particularly on read-intensive analytical processing workloads. In fact, column-stores are more I/O efficient for read-only queries since they only have to read from disk (or from memory) those attributes accessed by a query [25].

B-tree indexes are the most popular method for accelerating search operations. They are designed initially for transactional data (where any index on data must be updated quickly as data records are modified, and query results are usually limited in number of records) and fail to meet requirements of modern data analysis, such as interactive analysis over large volume of collected traffic data. Such queries return thousands of records that with b-trees would require a large number of tree-branching operations that use slow pointer chases in memory and random disk access, thus taking a long time. Many popular indexing techniques such as hash indexes, have similar shortcomings. Considering the peculiarity of network monitoring data where flow records are read-only and several flow fields have very few unique values, as of today the best indexing method is a bitmap index [33]. These indexes use bit arrays (commonly called bitmaps) and answer queries by performing bitwise logical operations on these bitmaps. For tasks that demand the fastest possible query processing speed, bitmap indexes perform extremely well because the intersection between the search results on each variable is a simple AND operation over the resulting bitmaps [22].

Seen that column-oriented databases with bitmap indexes provide better performance compared to relational databases, the authors explored their use in the field of flow monitoring. Hence they have designed a system based on this technology able to efficiently handle flow records. The main requirements of this development work include:

- Ability to save flow records on disk with minimal overhead allowing no-loss on-the-fly flow-to-disk storage, as it happens with tools based on raw files.
- Compact data storage for limiting disk usage hence enable users to store months of flow records on a cheap hard-disk with no need to use costly storage systems.
- Stored data must be immutable (i.e. once it has been saved it cannot be modified/ deleted) as this is a key feature for billing and security systems where non-repudiation is mandatory.
- Ability to perform efficiently on network storage such as NFS (Network File System).

- Simple data archive structure in order to move ancient data on off-line storage systems without having to use complex data partitioning solutions.
- Avoid complex architectures [34], hard to maintain and operate, by developing a simple tool that can be potentially used by all network administrators.
- On tens of millions of records:
  - Sub-second search time when performing cardinality searches (e.g. count the number or records that satisfy a certain criteria). This is a requirement for exploring data in real-time and implementing interactive drill-down data search.
  - Sub-minute search time when extracting records matching a certain criteria (e.g. top X hosts and their total traffic on TCP port Y).
- Feature rich query language as SQL with the ability to sort, join, and aggregate data while perform mathematical operations on columns (e.g. sum, average, min/max, variance, median, distinct), necessary to perform complex statistics on flows.

The following chapters covers the design and implementation of an extension to *nProbe* [35], an open-source probe and flow collector, that allows flow records to be stored on disk using a column-oriented database with an efficient compressed bitmap indexing technology. Finally the nProbe implementation performance is evaluated and positioned against similar tools previously listed.

3 Architecture and Implementation

nProbe is an open-source NetFlow probe that supports both NetFlow and sFlow collection, as well as flow conversion between versions (for instance convert v5 to v9 flows).

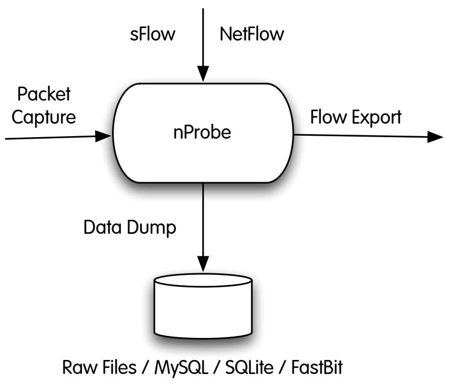

Fig. 1. nProbe Flow Record Collection and Export Architecture

It fully supports the NetFlow v9/IPFIX so it has the ability to specify dynamic flow templates (i.e. it supports flexible netflow) that are configured when the tool is started. nProbe features flow collection and storage, both on raw files and relational databases such as MySQL and SQLite. Support of relational databases has always been controversial as nProbe users appreciated the ability to query flow records using SQL, but at the same time flow dump to database is usually activated only for small sites. The reason is that enabling database support could lead to flow records loss due to the database processing overhead. This is mostly caused by network latency and multi-user database access, slow-down caused by table indexes update during data insertion, and poor database performance while searching records during data insertion. Databases offer mechanisms for mitigating some of the above issues, including data insertion in batch mode instead of realtime, transaction disabling, and definition of tables with no indexes for avoiding the overhead of indexes update.

In order to overcome the limitations of existing flow-management systems, the authors decided to explore the use of column-based databases by implementing an extension to nProbe that allows flows to be stored on disk using FastBit [29]. More precisely, FastBit is not a database but a C++ library that implements efficient bitmap indexing methods. Data is represented as tables with rows and columns. A large table may be partitioned into many data partitions and each of them is stored on a distinct directory, with each column stored as a separated file in raw binary form. The name of the data file is the name of the column. In each data partition there is an extra file named -part.txt that contains metadata information such as the name of the partition, and column names. Each column contains data stored in an uncompressed form, so its size is the same size of a raw file dump. Columns can accept data of 1, 2, 4, and 8 bytes long. Data longer than 8 bytes needs to be split across two or more columns. Compressed bitmap indexes are stored in separate files whose name is the name of the column with the .idx suffix. This means that each column typically has two files: one file contains data and the other the index. Indexes can be created on data “as stored on disk” or on reordered data. This is a main difference with respect to conventional databases. In fact it is possible to first reorder data, column by column, so that bitmap indexes are built on reordered data. Please note that reordering does not affect queries results (i.e. rows data is not mixed when columns are reordered), but it just improves index size and query speed. Data insert and query facilities is performed by means of library calls or using a subset of SQL, natively supported by the library. In FastBit the SELECT clause can only contain a list of column names and some functions that include AVG, MIN, MAX, SUM, and DISTINCT. Each function can only take a column name as its argument. The WHERE clause is a set of range conditions joined together with logical operators such as AND, OR, XOR, and NOT. The clauses GROUP BY, ORDER BY, LIMIT and the operators IN, BETWEEN and LIKE can also be applied to queries. FastBit actually does not support advanced SQL functionalities such as nested queries, and neither operators such as UNION, HAVING, or functions like FIRST, LAST, NOW, and FORMAT.

nProbe creates FastBit partitions depending on the flow templates being configured (probe mode) or read from incoming flows (collector mode), with columns having the same size as the the netflow element it contains. Users can configure partition duration (in minutes) at runtime and when a partition reaches its maximum duration, a new one is automatically created. Partition names are created on a tree fashion (e.g. <base directory>/year/month/day/hour/minute). Similar to [36], authors have developed facilities for rotating partitions hence limiting disk space usage while preserving their structure. No FastBit specific configuration is necessary as nProbe knows the flow format, and then it automatically creates partitions and columns. Datatypes longer than 64 bit as IPv6 addresses are transparently split onto two FastBit columns. Flow records are not saved individually on disk, but for efficiency reasons they are dumped in blocks of 4096 records. Users can decide to build indexes on all or only on a few selected columns, this in order to save space creating indexes for columns that will never be used in queries. If while executing a query FastBit does not find an index for a key column, it will build the index for such column on the fly, prior to execute the query. For efficiency reasons, the authors have decided that indexes are not built at every data dump but when a partition is completed (e.g. the partition duration time has elapsed). This happens because building indexes on reordered data is more efficient (both in terms of disk usage and query response time) than building them on data on the same order as it has been saved on disk. The drawback of this design choice is that queries can use indexes only once they have been built hence the partition is completely dumped on disk. On the other hand, flow records can be dumped at full speed with no index-build overhead. Thus, not considering flow receive/decoding overhead, it is possible to save on disk more than one million flow records/sec on a standard Serial ATA (SATA) disk. Column indexes are completely loaded into memory during searches, thus it imposes a limit on the partition size also limited by FastBit to 2^{32} records. Hence it is wise to avoid creating large partitions, but at the same time the creation of too many small partitions must also be avoided, as this will result in many files created on disk and the overhead of accessing them (open, close and file seek time) can dominate the data analysis time. A good compromise is to have partitions that either last a fixed amount of time (e.g. 5 minutes of flow records) or that have a maximum number of records. Typically, for a machine with a few GB of memory, FastBit developers recommend data partition containing between 1 million and 100 million records.

Conceptually a FastBit partition is similar to a table on a relational database, thus when a query is spread across several partitions, it is necessary to merge results and to collapse them when using the DISTINCT SQL clause. This task is not performed by FastBit but it is delegated to utilities developed by the authors:

- *fbmerge*: tool for merging several FastBit partitions into a single one. This tool, now part of the FastBit distribution, is useful when small fine grained partitions need to be aggregated into a larger one. For instance if nProbe is configured to create 'one minute' partitions, at the end of the hour all of them can be aggregated into a 'one hour' partition. This allows the number of column files hence the number of disk i-nodes to be reduced a lot, very useful on large disks containing many days/months of collected records.
- *fbquery*: tool that allows queries to be performed on partitions. It supports SQL-like syntax for querying data and implements on top of FastBit useful facilities such as:
  - Aggregation of similar results, data sort, and result set limitation (same as MySQL LIMIT).
  - Search recursively on nested directories so that a single directory containing several partitions can be searched in one shot. This is useful for instance when nProbe has dumped 5 minutes long partitions, and users want to search on the last hour so that various partitions need to be read by fbquery.
  - Data dump on several formats such as CSV, XML, and plain text. Data format is based on the metadata information produced by nProbe, thus partition columns are printed according to its native representation (e.g. an IPV4_DST_ADDR is printed as dot-separated IPv4 address and not as a 32 bit unsigned integer).
  - Scriptability using the Python language for combining several queries or creating HTML pages for rendering data on a web browser.

In a nutshell, the authors have used the FastBit library for creating an efficient flow collection and storage system. As the library was not designed for handling network flows, the authors have implemented some missing features that are a prerequisite for creating comprehensive network traffic reports. The following section evaluates the performance of the proposed solution, compares it against relational databases, and validates it on two large networks. This is to demonstrate that nProbe with FastBit is a mature solution that can be used on a production environment.

4 Validation and Performance Evaluation

In order to evaluate the FastBit performance, nProbe has been deployed in two different environments:

- Medium ISP: Bolig:net A/S
  The average backbone traffic is around 250 Mbit/sec (about 40K pps). The traffic is mirrored onto a Linux PC (Linux Fedora Core 8 32 bit, Dual Core Pentium D 3.0 GHz, 1 GB of RAM, two SATA III disks configured with RAID 1) that runs

nProbe in probe mode. nProbe computes the flows (NetFlow v9 bi-directional format with 5 minutes maximum flow duration) and saves flow records on disk using FastBit. Each FastBit partition stores one hour of traffic, and in average the probe produces 36 million flow records/day. Before deploying nProbe, records were collected and stored on a MySQL database.

- Large ISP: British Telecom
  nProbe is used in collector mode. It receives flow records from 10 peering routers, with peak flow export of 85 K flow records/sec with no flow loss. Each month the total amount of record exceeds 4 TB of disk space. The application server has dual quad-core Intel processors with 24 GB of memory, running Ubuntu Linux 9.10 64 bit, and is used to carry out queries on the data stored on an NFS server by the Collection server. The Netflow collection server has a single quad-core Intel processor and 8 GB of memory, running Ubuntu Linux 9.10 64 bit, and stores the fastbit data to the NFS server. Each FastBit partition stores 60 minutes of traffic that occupy about 5.8 GB of disk space when indexed. Before deploying nProbe, flow records were collected using nfdump.

The goal of these two setups is to both validate nProbe with FastBit on two different setups and compare the results with the solutions previously used. The idea is to compare a regional with a country-wide ISP, and verify if the proposed solution can be effectively used in both scenarios. Being the code open-source, it is also important to verify that this work is efficient when used on standard PCs (contrary to solutions based on costly clusters or server farms mostly used in Telco environments) as this is the most common scenario for many open-source users.

4.1 FastBit vs Relational Databases

The goal of this test is to compare the performance of FastBit with respect to MySQL (version 5.1.40 64 bit), a popular relational database. As the host running nProbe is a critical machine, in order to not interfere with the collection process, two days worth of traffic was dumped in FastBit format, and then transferred to a Core2Duo 3.06 GHz Apple iMac running MacOS 10.6.2. Moving FastBit partitions across machines running different operating systems and word length (one is 32, the other is 64 bit) has not required any data conversion as FastBit transparently takes care of differences among various architectures. This is a good feature as collector hosts can be based on different operating systems and technology. In order to evaluate how FastBit partition size affects the search speed, hourly partitions have been merged into a single daily directory. In order to compare both approaches, five queries have been defined:

- Q1: SELECT COUNT(*), SUM(PKTS), SUM(BYTES) FROM NETFLOW

- Q2: SELECT COUNT(*) FROM NETFLOW WHERE L4_SRC_PORT=80 OR L4_DST_PORT=80
- Q3: SELECT COUNT(*) FROM NETFLOW GROUP BY IPV4_SRC_ADDR
- Q4: SELECT IPV4_SRC_ADDR, SUM(PKTS), SUM(BYTES) AS s FROM NETFLOW GROUP BY IPV4_SRC_ADDR ORDER BY s DESC LIMIT 1,5
- Q5: SELECT IPV4_SRC_ADDR, L4_SRC_PORT, IPV4_DST_ADDR, L4_DST_PORT, PROTOCOL, COUNT(*), SUM(PKTS), SUM(BYTES) FROM NETFLOW WHERE L4_SRC_PORT=80 OR L4_DST_PORT=80 GROUP BY IPV4_SRC_ADDR, L4_SRC_PORT, IPV4_DST_ADDR, L4_DST_PORT, PROTOCOL

FastBit partitions have been queried using the fbquery tool with appropriate command line parameters. All MySQL tests have been performed on the same machine with no network communications between client and server (i.e. MySQL client and server communicate using a Unix socket). In order to evaluate the influence of MySQL indexes on queries, the same test has been repeated with and without indexes. Tests were performed on 68 million flow records containing a subset of all NetFlow fields (IP source/destination, port source/destination, protocol, begin/end time). The following table compares the disk space used by MySQL and FastBit. In the case of FastBit, indexes have been computed on all columns.

MySQL	**No/With Indexes**	1.9 / 4.2
FastBit	**Daily Partition (no/with Indexes)**	1.9 / 3.4
	Hourly Partition (no/with Indexes)	1.9 / 3.9

Table 1. FastBit vs MySQL Disk Usage (results are in GB)

Query	**MySQL**		**Daily Partitions**		**Hourly Partitions**	
	No Index	**With Indexes**	**No Cache**	**Cached**	**No Cache**	**Cached**
Q1	20.8	22.6	12.8	5.86	10	5.6
Q2	23.4	69	0.3	0.29	1.5	0.5
Q3	796	971	17.6	14.6	32.9	12.5

Query	MySQL		Daily Partitions		Hourly Partitions	
	No Index	With Indexes	No Cache	Cached	No Cache	Cached
Q4	1033	1341	62	57.2	55.7	48.2
Q5	1754	2257	44.5	28.1	47.3	30.7

Table 2. FastBit vs MySQL Query Speed (results are in seconds)

The test outcome has demonstrated that FastBit takes approximately the same disk space as MySQL in terms or raw data, whereas MySQL indexes are much larger. Merging FastBit partitions does not usually improve the search speed, but instead queries on merged data requires more memory, as FastBit loads a larger index.

The size/duration of a partition mostly depends on the application that will access data. Having small partitions (e.g. 1 or 5 minutes long) makes sense for interactive data exploration where drill-down operations are common. In this case, having small partitions means that the FastBit index would also be small, resulting in faster operations and less memory used. On the other hand, querying data on a long period using small partitions requires fbquery to read several small indexes instead of a single one that is inefficient on standard disks (i.e. non solid-state drive) due to disk seek time. In addition, a side effect of multi-partitions is that fbquery need to merge results produced on each partition, this without relying on FastBit. Note that the use of large partitions has drawbacks on searches, as indexes cannot be built on the them until they have been completely dumped. For this reason, if nProbe saves flow records on a large one day long partition, it means that queries on the current day must be performed without indexes as the partition has not completely dumped yet. In a nutshell there is not a single rule for defining partition duration; in general the partition granularity should be as close as possible to the expected query granularity. Authors suggest to use partitions lasting from 1 to 5 minutes in order to have quick searches even on partitions being written (i.e. on most recent data), and then daily merge partitions using fbmerge. This to avoid exhausting disk i-nodes with index files, and efficiently perform searches on past data without accessing too many files.

In terms of query performance FastBit is not at all comparable with MySQL:

- Queries that only require access to indexes take less than a second, regardless of the query type.
- Queries that require data access are at least an order of magnitude faster than on MySQL but always complete within a minute.
- Index creation time on MySQL takes many minutes and it prevents it using in real life when importing data in (near-)realtime, also considering that they take a significant amount of disk space. Indexes on MySQL do not speed up queries,

contrary to FastBit, as query time using indexes takes longer when compared to the same query on unindexed data.

- Disk speed is an important factor for accelerating queries. In fact running the same test twice with data already cached in memory, it significantly decreases the query speed. The use of RAID 0 has demonstrated that the performance speed has been improved.

4.2 FastBit vs Raw Files

The goal of this test is to compare FastBit with a popular open-source collection tool named nfdump. Tests have been performed on a large network with TB of collected flow data per month. Although nfdump performs well when it comes to flow collection, its performance is sub-optimal during query time when using large data sets. One of the main concerns of the network operators is that with nfdump queries take a long amount of time, so they often need to be run overnight before producing results. An explanation of this behavior is that nfdump does not index data, so searching on a large time span means reading all raw data that was received over that period, and in this setup means GBs (if not TBs) of records. Using FastBit the average speed improvement is in the order of 20:1. From the operator's point of view this means that queries can last a reasonable amount of time. For instance, a query written in SQL as ‘SELECT IPV4_SRC_ADDR, L4_SRC_PORT, IPV4_DST_ADDR, L4_DST_PORT, PROTOCOL FROM NETFLOW WHERE IPV4_SRC_ADDR=X OR IPV4_DST_ADDR=X’ on 19 GB of data that contain 14 hours of collected flow records, takes about 45 seconds with FastBit which is major improvement with respect to nfdump, which takes about 1500 seconds (25 minutes) to complete the same query. As nfdump does not use any index, its execution time is dominated by the time needed to sequentially read the binary data. This means that: query time = (time to sequentially read the raw data) + (record filtering time). The time needed to filter records is usually very little as nfdump is fast enough, and also because the complexity of filters, whose syntax is similar to BPF [37] filters, is usually limited. This means that in nfdump the query time is basically the time needed to sequentially read the raw data. The previous query validates this hypothesis: 1500 seconds to read 19 GB of data means that the average reading speed is about 12.6 MB/sec, that is the typical speed of a SATA drive. For this reason, this section does not list the same tests as in section 4.1, because the query time of nfdump is mostly proportional to the amount of data to read [20]; hence with some simple math it is possible to compute the expected nfdump response time. Also note that the nfdump query language is not SQL-like, therefore it is not possible to make a one-to-one comparison with FastBit and MySQL.

As flow records take a large amount of disk space, it is likely that they will be stored on a SAN (Storage Area Network). When the storage is directly attached to the host by means of fast communication links such as InfiniBand and FibreChannel, the system does not see any speed degradation when compared with a directly attached SATA disk. The authors decided to study how the use of network file systems such as NFS affects the query results. A simple model for the time needed to read γ bytes is $t = \alpha + \beta * \gamma$, where α represents the disk access latency and β is the throughput. NFS typically increases α but not β as the network speed is typically higher than disk read speed. In the case of nfdump the data is read sequentially, whereas on FastBit the raw data is accessed based on indexes. Thus FastBit requires a small number of read operations which have to pay α multiple times. However this extra cost is in milliseconds, so it does not alter the overall comparison. This behavior has been tested repeating some queries of 4.1, and demonstrating that the use of NFS marginally affects the total query time.

4.3 FastBit Scalability

The tests have shown that the use of FastBit offers advantages with respect to both relational databases and raw files-based solutions. In order to understand nProbe scalability when used with FastBit, it is necessary to split flow collection from flow query. As stated in section 3, the index creation happens when the partition has been dumped on disk, hence the dump speed to disk is basically the speed of the hard drive where, in the case of SATA disks, it exceeds 1 million flow records/sec. As shown in 4.2, a large ISP network produces less than 100’000 flow records/sec, this means that FastBit introduces no bottleneck in flow collection and dump. Flow query requires disk access, therefore the query process is mostly I/O bound. For every query, FastBit reads the whole index file of each column present in the WHERE clause. Then based on the index search, it reads if necessary (e.g. COUNT(*) does not require that) the column files containing real data by performing seeks on files in order to move to the offset where the index has found a data match. Thus a simple model for query response time is $\tau = \gamma + \varepsilon + \delta$, where γ represents the time needed to read all the column indexes present in the WHERE clause, ε is the time to read (if there is any) the matching rows data present in the SELECT clause, and δ is the processing overhead. In general δ is very limited with respect to γ and ε. As γ = (index size / disk speed), it takes no more than a couple of seconds. Instead δ can be pretty large if data is sparse, and several disk seeks are required. Note that δ can grow significantly depending on the query (e.g. in case of sorting large data sets), and that ε is zero for queries that count (e.g. compute the number of records on port X produced by host Y) or that use mathematical functions such as SUM (e.g. total number of bytes on port X).

5 Open Issues and Future Work

Tests on various FastBit configurations have shown that the disk is an important component that has a major impact on the whole system. The authors are planning to use SSD drives in order to see how query time is affected, in particular while accessing raw data records that require several disk seek operations.

One of the main limitations of FastBit is the lack of data compression, since it currently compresses only indexes. This is a feature that the authors are planning to add, as it allows disk space to be saved hence reduce the time needed to read the records. Using compression libraries such as QuickLZO, lzop, and FastLZ it should be possible to implement transparent de/compression while reducing disk space.

Another area of interest is the use of FastBit for indexing packets instead of flows. The authors have prototyped an application that parses pcap files and creates a FastBit partition based on various packet fields such as IP address, port, protocol, and flags, in addition to an extra column that contains the file id and packet offset inside the pcap file. Using a web interface built on top of fbquery, users can search packets matching the criteria and also retrieve the original packet contained in the original pcap files. Although this work is not rich in features when compared with specialized tools [36], it demonstrates that the use of bitmap indexes is also effective for handling packets and not just flow records.

The work described on this paper is the base for developing interactive data visualization tools based on FastBit partitions. Thanks to recent innovation in web 2.0, there are libraries such as the Google Visualization API that split visualization from data. Currently, the authors are extending nProbe adding an embedded web server that can make FastBit queries on the fly and return query results in JSON format [38]. The idea is to create an interactive query system that can visualize both tabular data (e.g. flow information) and graphs (e.g. average number of flow records on port X over the last hour) by means of FastBit queries. This way the user does not have to interact with FastBit tools at all, and can focus on data exploration.

6 Final Remarks

The use of nProbe with FastBit is a major step ahead when compared to state-of-the-art tools based on both relational databases and raw data dump. When searching data on datasets of a few million records the query time is limited to a few seconds in the worst case, whereas queries that just use indexes are completed within a second. The consequence of this major speed improvement is that it is now possible to query data in real time and avoid periodically updating costly counters, as their value can be computed on-demand using bitmap indexes. Finally this work paves the way to the creation of new monitoring tools on large data sets that can interactively analyze traffic data in near-realtime, contrary to what usually happens with most tools available today.

Availability. This work is distributed under the GNU GPL license and is available at the ntop home page http://www.ntop.org/nProbe.html.

Acknowledgments. The authors would like to thank K. John Wu <kwu@lbl.gov> for his help and support while using the FastBit library, Anders Kjærgaard Jørgensen <akj@bolignet.dk> for his support during the validation of this work, and Cristian Morariu <morariu@ifi.unizh.ch> for his suggestions during MySQL tests.

References

1. B. Claise, NetFlow Services Export Version 9, RFC 3954, (2004).
2. P. Phaal and others, InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks, RFC 3176, (2001).
3. J. Quittek, and others, Requirements for IP Flow Information Export (IPFIX), RFC 3917, (2004).
4. H. Haddadi and others, Revisiting the Issues on Netflow Sample and Export Performance, ChinaCom 2008, 442-446, (2008).
5. N. Duffield and others, Properties and Statistics from Sampled Packet Streams, Proc. ACM SIGCOMM IMW' 02, (2002).
6. C. Estan and others, Building a better NetFlow, Proc. of the ACM SIGCOMM Conference, (2004).
7. S. Chakchai, A Survey of Network Traffic Monitoring and Analysis Tools, (2006).
8. C. Ning and X. Tong-Ge, Study on NetFlow-based Network Traffic Data Collection and Storage, Application Research of Computers, Vol. 25, no. 2, (2008).
9. F. Reiss and others, Enabling Real-Time Querying of Live and Historical Stream Data, Proc. of 19th Intl. Conference on Scientific and Statistical Database Management, (2007).
10. P. Haag, Watch your Flows with NfSen and NfDump, 50th RIPE Meeting, (2005).
11. M. Fullmer and S. Roming, The OSU Flow-tools Package and Cisco NetFlow Logs, Proc. of 14th USENIX Lisa Conference, (2000).
12. D. Plonka, FlowScan: A Network Traffic Flow Reporting and Visualization Tool, Proc. of 14th USENIX Lisa Conference, (2000).
13. A. Øslebø, Stager A Web Based Application for Presenting Network Statistics, Proc. of NOMS 2006, (2006).
14. C. Gates and others, More NetFlow Tools: For Performance and Security, Proc. 18th Systems Administration Conference (LISA), (2004).
15. J.P. Navarro and others, Combining Cisco NetFlow Exports with Relational Database Technology for Usage Statistics, Intrusion Detection and Network Forensics, Proc. 14th Systems Administration Conference (LISA), (2000).
16. P. Lucente, pmacct: a New Player in the Network Management Arena, RIPE 52 Meeting, (2006).
17. V. Marinov and J. Schönwälder, Design of an IP Flow Record Query Language, Proc. of AIMS 08, Springer LNCS 5127, (2008).

18. A. Sperrotto, Using SQL databases for flow processing, Joint EMANICS/IRTF-NMRG Workshop on Netflow/IPFIX Usage in Network Management, (2008).
19. R. Hofstede, Performance measurements of NfDump and MySQL and development of a SURFmap plug-in for NfSen, Bachlor assignement, University of Twente, (2009).
20. R. Hofstede and others, Comparison Between General and Specialized Information Sources When Handling Large Amounts of Network Data, Technical Report, University of Twente, (2009).
21. M. Siekkinen and others, InTraBase: Integrated Traffic Analysis Based on a Database Management System, Proc. of E2EMON '05, (2005).
22. K. Wu and others, A Lightning-Fast Index Drives Massive Data Analysis, SciDAC Review, (2009).
23. J. Oltsik, The Silent Explosion of Log Management, CNET News, http://news.cnet.com/8301-10784_3-9867563-7.html, (2008).
24. M. J. Turner and others, A DBMS for large statistical databases, Proc. of 5th VLDB Conference, (1979).
25. D. Abadi and others, Column-Stores vs. Row-Stores: How Different Are They Really?, Proc. of ACM SIGMOD '08, (2008).
26. O. Herrnstadt, Multiple Dimensioned Database Architecture, U.S. Patent application 20090193006, (2009).
27. D. Loshin, Gaining the Performance Edge Using a Column-Oriented Database Management System, White Paper, (2009).
28. K. Wu and others, Compressed Bitmap Indices for Efficient Query Processing, Technical Report LBNL-47807, (2001).
29. K. Wu and others, FastBit: Interactively Searching Massive Data, Proc. of SciDAC 2009, (2009).
30. E.W. Bethel and others, Accelerating Network Traffic Analytics Using Query-Driver Visualization, IEEE Symposium in Visual Analytics Science and Technology, (2006).
31. D. Abadi and others, Integrating Compression and Execution in Column-Oriented Database Systems, Proc. of 2006 ACM SIGMOD, (2006).
32. F. Otten, Evaluating Compression as an Enabler for Centralised Monitoring in a Next Generation Network, Proc. of SATNAC 2007, (2007).
33. V. Sharma, Bitmap Index vs. B-tree Index: Which and When?, Oracle Technology Network, (2005).
34. J. Chandrasekaran and others, TelegraphCQ: Continuous Dataflow Processing for an Uncertain World, Proc. of Conference on Innovative Data Systems Research, (2003).
35. L. Deri, nProbe: an Open Source NetFlow Probe for Gigabit Networks, Proc. of Terena TNC 2003, (2003).
36. P. Desnoyers and P. Shenoy. Hyperion: High Volume Stream Archival for Retrospective Querying, Proc. of 2007 USENIX Annual Technical Conference, (2007).
37. S. McCanne and V. Jacobson, The BSD Packet Filter: A New architecture for User-level Packet Capture, Proc. Winter '93 USENIX Conference, (1993).
38. D. Crockford, JSON: The fat-free alternative to XML, Proc. of XML 2006, (2006).