Collection and Exploration of Large Data

Why the use of FastBit is a major step ahead when compared with state of the art relational database tools based on relational databases.

What you will learn…

- Basics of building traffic monitoring applications
- Data Collection and Exploration

What you should know…

- A basic knowledge of architecture and implementation of traffic monitoring tools
- A basic knowledge of TCP/IP

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. In this article I would like to approach to the problem by using a bitmap database that allows to implementation of an efficient solution for both data collection and retrieval.

NetFlow and sFlow

NetFlow and sFlow are the current standards for building traffic monitoring applications. Both are based on the concept of a traffic probe (or agent in the sFlow parlance) that analyses network traffic and produces statistics, known as flows, which are delivered to a central data collector. As the number of flows can be pretty extremely high, both standards use sampling mechanisms in order to reduce the workload on bothof the probe and collectors. In sFlow the use of sampling mechanisms is native in the architecture so that it can be used on agents to effectively reduce the number of flows delivered to collectors. *This has a drawback in terms of result accuracy while providing results with quantifiable accuracy*. With NetFlow, the use of sampling (both on packets and flows) leads to inaccuracy and this means that flows sampling is very seldom used in NetFlow hence there is no obvious mechanism for reducing the number of flows records while preserving accuracy. For these reasons, network operators usually avoid sampling data hence have to face with the problem of collecting and analyzing a large number of flows that is often solved using a flow collector that stores data on a SQL-based relational database or on disk in raw format for maximum collection speed. Both approaches have pros and cons; in general SQL-based solutions allows 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 is to overcome the limitations of existing solutions and create a new generation of a flow collection and storage architecture that exploits state-of-the-art indexing and querying technologies. In the following I would like to describe the design and implementation of nProbe , an open-source probe and flow collector, that allows flows to be stored on disk using the FastBit database.

Architecture and Implementation

nProbe is an open-source NetFlow probe that also supports both NetFlow and sFlow collection and, flow conversion between version (for instancei.e. convert v5 to v9 flows). It fully supports the NetFlow v9 specification so giving it has the ability to specify flow templates (i.e. it supports flexible netflow) that are configured at runtime when the tool is started (Figure 1).

When used as probe and collector, nProbe supports flow collection and storage to either raw files or relational databases such as MySQL and SQLite. Support of relational databases has always been controversial as users appreciated the ability to search flows using a SQL interface, but at the same time flow dump to database is usually enable only realistic for small sites. The reason is that enabling database support could lead to the loss of flows due to the database overhead. There are multiple reasons that contribute to this behavior and in particularincluding:

- Network latency and multi-user database access for network-based databases.
- Use of SQL that requires flow information to be converted into text that is then interpreted by the database, instead of using an API for directly writing into the database.
- Slow-down caused by table indexes update during data insertion.
- Poor database performance when searching data during data insert.

Databases offer mechanisms for partially avoiding some of the above issues, which includinge:

- Data insert in batch mode instead of doing it in real time.
- Avoid network communications by using file-based databases.
- Disable database transactions.
- Use efficient table format optimized for large data tables.
- Not defining tables indexes therefore avoiding the overhead of index updates, though usually results in slower data search time.

Other database limitations include the complexity of handling large databases containing weeks of data, and purging old data while still accommodating new flow records. Many developers partition the database often creating a table per day that is then dropped when no longer needed.

The use of file-based databases such as SQLite offer a few advantages with respect
to networked relational databases, as:

- It is possible to periodically create a new database (e.g. one database per hour) for storing flows received during that hour, this is in order to avoid creating large databases.
- According to some tests performed, the flow insert throughput is better than networked-databases but still slower than raw flow dump.
- In order to both overcome the limitations of relational databases, and avoid raw flow dump due to limited query facilities, I decided to investigate the use of column-based databases and in particular, of FastBit .

Validation and Performance Evaluation

I have used the FastBit library for creating an efficient flow collection and storage system. This is to demonstrate that nProbe with FastBit is a mature solution that can be used on in a production environment. In order to evaluate the FastBit's performance, nProbe has been deployed in two different environments:

Medium ISPs

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, Kernel 2.6.23, 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 and saves them on disk using FastBit. In order to reduce the number of flows, the probe is configured to save flows in NetFlow v9 bi-directional format with maximum flow duration of 5 minutes. In average the probe generates 36 million flows/day. Each FastBit partition stores one hour of traffic. Before deploying nProbe, flows were collected and stored in a MySQL database.

Large ISPs

nProbe is used in collector mode. It receives flows from 8 peering routers, with peak flow export of 85 K flows/sec. The collection server is a fast machine with 8 GB of memory, running Ubuntu Linux 9.10 server 64 bit. Each FastBit partition stores five minutes of traffic that occupy about 5.8 GB of disk space. A second server running Ubuntu Linux 9.10 server 64bit and 24 GB of memory is used to query the flow data. The FfastBbit partitions are saved to a NFS mount on a local storage server. Before deploying

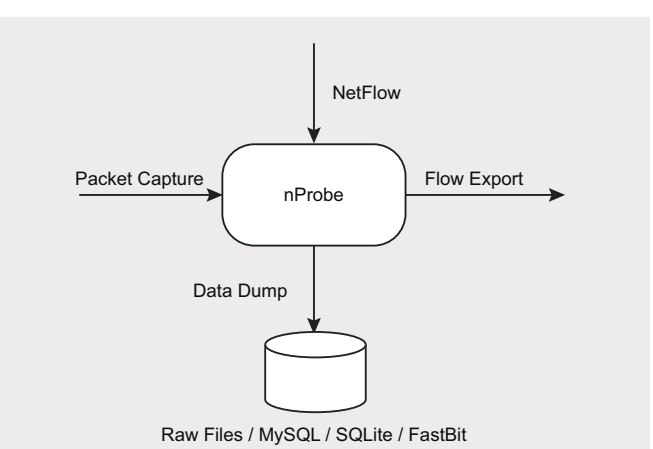

Figure 1.

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

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

nProbe, flows were collected using nfdump and each month the total amount of flow dumps exceeds 4 TB of disk space. The goal of these two setups is to both validate nProbe with FastBit on two different setups and compare the results with the solution previously used.

FastBit vs Relational Databases

Let´s 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 not to interfere with the collection process, two days worth of traffic was dumped in FastBit format, and then transfered 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. This is a good feature as over-time collector hosts can be based on various operating systems and technology; hence flow archives can be used immediately without any data conversion is a desirable feature. In order to evaluate how FastBit partition size affects the search speed, hourly partitions have been merged into a single daily partition. In order to compare both approaches, five queries can be 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`

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

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
Q4	1033	1341	62	57.2	55.7	48.2
Q5	1754	2257	44.5	28.1	47.3	30.7

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. In order to evaluate the influence of MySQL indexes on queries, the same test has been repeated with and without indexes.

Data used for testing washave been captured on Oct 12th and 13th (~68 million flows) and contained a subset of NetFlow fields (IP source/destination, port source/destination, protocol, begin/end time). The table below compares the disk space used by MySQL and FastBit. In the case of FastBit, indexes have been computed on all columns.

Merging FastBit partitions does not usually improve the search speed but instead queries on merged data requires more memory as FastBit has to load a larger index in memory. In terms of query performance, FastBit is far superior compared with MySQL as shown in Table 2:

- Queries that require access only 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 that on MySQL.
- Index creation time on MySQL takes many minutes and it prevents its use in real life when importing data in (near-)realtime, and also indexes also take a significant amount of disk space.
- Indexes on MySQL do not speed up queries, contrary to FastBit.
- Disk speed is an important factor for accelerating queries. In fact running the same test twice with data already cached in memory, significantly decreases the query speed. The use of RAID 0 has demonstrated that the performance speed has been improved.

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. I am planning to explore the use of solid-state drives in order to see if the overall performance can benefit from it.performance increases.

A main limitation of FastBit is the lack of data compression as it currently compresses only indexes but not data. This is a feature is planned to add, as it allows disk space to be saved hence to reducereducing the time needed to read the data.

This article is the base for developing interactive data visualization tools based on FastBit partitions. Thanks to recent innovations in web 2.0, there are libraries such as the Google Visualization API that allow separating data rendering from data source. Currently we are extending nProbe adding an embedded web server that can make FastBit queries on the fly and return query results in JSON format. 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 flows on port X over the last hour) by performing FastBit queries. This way the user does not have to interact with FastBit tools at all, and can focus on data exploration.

Final Remarks

The use of FastBit is a major step ahead when compared with state of the art tools based on both relational databases and raw data dumps. 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 updating costly counters every second, as using bitmap indexes it is possible to produce the same information when necessary. 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-real time, 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*. The nBox appliance embedded withing a pre-installed ntop and nProbe software can be requested at *www.wuerth-phoenix.com/nbox*.

LUCA DERI, FOUNDER OF NTOP

Luca Deri was born in 1968. Although he was far too young to remember, the keywords of that year were freedom, equality, free thinking, revolution. In early 70s many free radio stations had birth here in Italy because their young creators wanted to have a way for spreading their thoughts, ideas, emotions and tell the world that they were alive 'n kickin'. The Internet today represents for him what free radio represented in the 70s. He wrote his PhD on Component-based Architecture for Open, Independently Extensible Distributed Systems. Luca Deri is the founder of Ntop.