nProbe with FastBit Database: an Innovative Flows Storage Solution

nProbe, acronym for NetFlow probe, is an open-source probe that supports both NetFlow and sFlow collection, as well as flow conversion between versions (for instance convert v5 to v9 flows). It fully supports the NetFlow v9/IPFIX so it has the ability to specify dynamic flow templates that are configured when the tool is started.

nProbe has been designed to keep up with Gigabit speeds on commodity hardware and it can be used for capturing packets and analyzing networks at full speed with no (or very moderate) packet loss using PF_RING[1].

Each captured packet is analyzed, associated to a flow record, and stored onto a hash. Periodically, the hash is analyzed and expired flows are emitted and exported to the specified collectors. nProbe is fully inter-operable with commercial collectors such as Fluke, Cisco NetFlow Collector, Dartware, AdventNet, Arbor Networks, Plixer, NetFlow Auditor, SolarWinds Orion NTA. Exported flows can also be analyzed using open source tools such as ntop.

nProbe Storage System

When nProbe is used as probe and collector, it supports 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 could lead to flow records loss due to the database-processing overhead. On the contrary, the speed advantage of dumping flow records in raw format is paid at each search operation in terms of amount of data to read. Furthermore, the query language that can be used is limited when compared to SQL facilities.

In order to overcome the limitations of existing flow-management systems, an extension of nProbe has been developed. This allows flow records to be stored on disk, using an innovative column-oriented database with an efficient compressed bitmap indexing technology named FastBit.

A column-oriented database stores its content by column, rather than by row (known as vertical organization). On 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. Furthermore, for tasks that demand the fastest possible query processing speed, bitmap indexes perform extremely well. These because the intersection between the search results on each variable is a simple AND operation over the resulting bitmaps.

Conceptually FastBit is a database where data is represented as tables with rows and columns allowing storing and retrieving of data, designed primarily to answer queries efficiently.

1 More details about PF_RING can be found here: www.ntop.org/PF_RING.html

The extended nProbe creates FastBit partitions depending on the flow templates being configured (in probe mode) or read from incoming flows (in collector mode), with columns having the same size and the same name as the NetFlow element it contains.

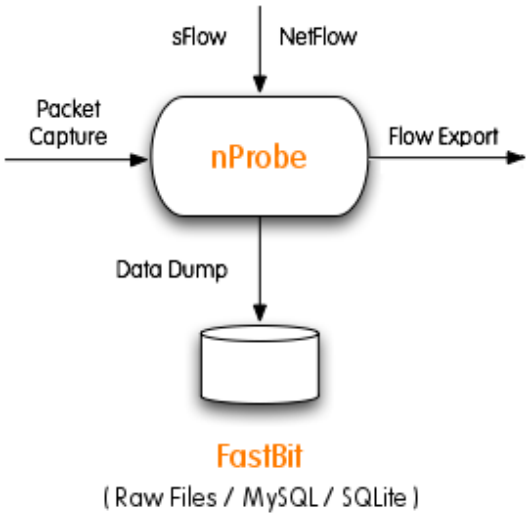

Fig. 1 New nProbe Flow Record Collection and Export Architecture

It is not necessary to configure FastBit in a specific way as nProbe knows the flow format, and then it automatically creates partitions and columns. A large table may be partitioned into many data partitions and each of them are stored on a distinct directory, with each column stored as a separated file in raw binary form.

Users can configure partition duration (in minutes) at runtime and when a partition reaches its maximum duration, a new one will be automatically created. Partition names are created on a tree fashion (e.g. `<dir>/year/month/day/hour/minute`) and in each data partition there is an extra file, named `-part.txt`, that contains meta data information such as the name of the partition, and column names. In addition, there have been developed facilities for rotating partitions hence limiting disk space usage while preserving their structure.

The use of nProbe with FastBit is a major step ahead if compared to state-of-the-art tools based on both relational databases and raw data dump. 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.

Additional advantages of this technology are listed below:

- 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 to limit disk usage as this enables users to store months of flow records on a cheap hard-disk with no need to use expensive storage systems.

- Simple data archive structure in order to move ancient data on off-line storage systems, without having to use complex data partitioning solutions.

- 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) and sub-minute search time when extracting records matching a certain criteria (e.g. top X hosts and their total traffic on TCP port Y).

nProbe Command Line Options

Only the additional options that can be used to configure nProbe to store flow records using the new FastBit format are listed below.

`--fastbit <dir>`	Pathname of base directory where FastBit files will be created. Partition names are created on a tree fashion (e.g. `<dir>/year/month/day/hour/minute`)
`--fastbit-rotation <mins>`	Every `<mins>` minutes a new FastBit sub-directory is created so that each directory contains at most `<mins>` minutes. Default: 5 minutes. Please read the note below.
`--fastbit-template <template>`	Fields that will be dumped on FastBit partitions. Its syntax is the same as the `-T` flag (see the appendix for further information about the NetFlow fields). The selected fields must start with a `%` symbol and must be separated by a space. If this flag is not specified, all the specified flow elements (`-T`) will be dumped.
`--fastbit-index <template>`	Index each directory containing FastBit files as soon as the directory has been dumped. The flow template specifies which columns will be indexed. Its syntax is the same as the `-T` flag (see the appendix for further information about the NetFlow fields). This option requires that the **`fbindex`** application is installed or built. If this flag is not specified, all columns will be indexed.
`--fastbit-exec <cmd>`	Executes the specified command `<cmd>` after a directory has been dumped (and optionally indexed). The command must take an argument that is the path to the directory just dumped.

Note:
It is wise to avoid creating large partitions, but at the same time the creation of too many small partitions must also be avoided. Thus 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.

nProbe Examples

Some examples of command line options and output messages of the latest version of nProbe configured to use the FastBit database are shown below.

A simple example of nProbe configured to dump flow records in the temporary directory `/tmp/fastbit/` with a rotation period of 10 minutes:

```
nprobe -n none -i eth0 --fastbit /tmp/fastbit/ --fastbit-rotation 10 --fastbit-template
"%IPV4_SRC_ADDR %IPV4_DST_ADDR %IN_PKTS %IN_BYTES %OUT_PKTS %OUT_BYTES %FIRST_SWITCHED
%LAST_SWITCHED %L4_SRC_PORT %L4_DST_PORT %TCP_FLAGS %PROTOCOL"
```

An example of nProbe configured to dump flow records in the temporary directory and to index only the "`IPV4_SRC_ADDR %IPV4_DST_ADDR`" columns after each directory that has been dumped:

```
nprobe -n none -i eth0 --fastbit /tmp/fastbit/ --fastbit-template "%IPV4_SRC_ADDR
%IPV4_DST_ADDR %IN_PKTS %IN_BYTES %OUT_PKTS %OUT_BYTES %FIRST_SWITCHED %LAST_SWITCHED
%L4_SRC_PORT %L4_DST_PORT" --fastbit-index "%IPV4_SRC_ADDR %IPV4_DST_ADDR"
```

An example of nProbe configured to dump flow records in the temporary directory `/tmp/fastbit/` and to execute a specified command after each directory that has been dumped. In this simple example, the command to be run is "`ls`":

```
nprobe -n none -i eth0 --fastbit /tmp/fastbit/ --fastbit-template "%IPV4_SRC_ADDR
%IPV4_DST_ADDR %IN_PKTS %IN_BYTES %OUT_PKTS %OUT_BYTES %FIRST_SWITCHED %LAST_SWITCHED
%L4_SRC_PORT %L4_DST_PORT" --fastbit-exec "ls"
```

Finally, an example of the output message of the latest version of nProbe with FastBit database configured and the list of possible files contained in a FastBit directory:

```
[nprobe.c:2266] Welcome to nprobe v.5.5 ($Revision: 1284 $) for i686-pc-linux-gnu
..
[fastbit/fastbit.c:172] Fastbit files will be saved in /tmp/fastbit/2010/02/13/15/00
[fastbit/fastbit.c:357] Successfully initialized FastBit
..
[nprobe.c:3411] Capturing packets from interface eth0
```

```
> ls /tmp/fastbit/2010/02/13/15/00
-part.txt
FIRST_SWITCHED
IN_BYTES
IN_PKTS
IPV4_DST_ADDR
IPV4_SRC_ADDR
L4_DST_PORT
L4_SRC_PORT
LAST_SWITCHED
OUT_BYTES
OUT_PKTS
PROTOCOL
TCP_FLAGS
```

FastBit-based tools: fbindex, fbmerge and fbquery

Data insert and query facilities are performed by means of FastBit library calls or using a subset of SQL, natively supported by the library. Due to the fact that the FastBit database was not designed to handle network flow records, have been developed three tools to implement some missing features that are a prerequisite to create comprehensive network traffic reports.

fbindex

This tool is used to index a FastBit partition. Indexes can be created on data "as stored on disk" or on reordered data. This is a main difference 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 just improves index size and query speed.

This tool can be used with nProbe (parameter `--fastbit-index`) to index a partition as soon as the directory has been dumped by nProbe.

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 FastBit does not find an index for a key column while executing a query, it will build the index for such a column on the fly.

```
Usage: fbindex [-s] [-c <column names>] [-t <threads>] -d <directory>
```

`-s`	Flag to reorder data before indexing
`-c <column names>`	Specify a list of column names separated by comma to index.
`-t <threads>`	Specify the number of threads to use during the indexing process. Default: 2
`-d <directory>`	Absolute or relative pathname of directory to index

fbmerge

This tool is used to merge several FastBit partitions into a single one. The mentioned 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 an "one hour" partition by means of `fbmerge`. 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.

```
Usage: fbmerge -i <input dir> -o <output dir> [-d]
```

`-i <input directory>`	Absolute or relative pathname of input directory
`-o <output directory>`	Absolute or relative pathname of output directory
`-d`	Flag to enable dump mode

fbquery

This tool has been developed to perform queries on FastBit partitions dumped by nProbe. `fbquery` is a command-line tool that supports SQL-like syntax to query data and that allows to implement on top of the FastBit library some useful facilities.

The queries accepted by `fbquery` are indicated in a simplified SQL format. A query is basically a SQL SELECT statement where the following clauses can be specified:

- an optional SELECT clause using the `fbquery` parameter "`--columns`"
- a mandatory FROM clause using the `fbquery` parameter "`--directory`"
- an optional WHERE clause using the `fbquery` parameter "`--query`"

According to the SQL standard all the keywords, the operators, the function names, and the names of the columns are not case sensitive. Moreover, the blank spaces around the parameters, around the operators or around column names are ignored.

The following paragraph shows the requirements of these three clauses implemented in `fbquery` and shows some examples of use.

```
Usage: fbquery [-c <columns>] -d <directory> [-q <query-conditions>]
               [-o <orderby-columns>] [-r] [-g <groupby-columns>]
               [-m <metadata-file>] [-D] [-L <i[,j]>] [-S <separator>]
               [-H] [-N] [-V] [-Q]
```

Information about the program	
`--help` `-h`	Shows help message and exit
`--version` `-v`	Shows `fbquery` version number and exit
Parameters for input data	
`--columns <column names>` `-c <column names>`	List of column names separated by comma and any of the four one-argument functions: `MIN()`, `MAX()`, `SUM()` and `AVG()`.
`--directory <directory>` `-d <directory>`	Pathname of directory containing a Fastbit partition on which execute the query. Multiple partitions can be defined using multiple `-d` flags.
`--query <conditions>` `-q <conditions>`	It specifies the conditions of the query with a set of range conditions joined together with logical operators. The supported logical operators are: `AND`, `OR`, `XOR` and `NOT`. The range conditions can be defined with the operators: `<`, `<=`, `>`, and `>=`.
`--orderby <column names>` `-o <column names>`	Used to sort the results in according to columns specified (can be specified more columns comma separated). By default, the values are sorted in ascending order.
`--reverse` `-r`	Sort values in descending order in according to columns in the ORDER BY clause (or in the SELECT clause, if no ORDER BY clause is specified).
`--groupby <column names>` `-g <column names>`	Used along with the aggregate functions like `SUM()` to provide means of grouping the result by certain columns.
`--metadata-file <file>` `-m <file>`	Pathname of the file with flow metadata that has been dumped by nProbe. (This file allows to `fbquery` to print results with a customized format)
`--distinct` `-D`	Retrieves only unique rows from the result set.

`--limit <i[,j]>` `-L <i[,j]>`	Limit query results to those that fall within a specified range. Can be used to show the first `i` number of results, or to show a range using `<i,j>`, where `i` is the starting point (the first record is 0) and `j` is the duration (how many records to display) .
`--separator <separator>` `-S <separator>`	Character to be used as CSV-separator between columns of results.
`--hide-header` `-H`	Hide the header line that contains the column names.
`--native-format` `-N`	Print each selected column with its native format.
Verboseness of the program	
`--verbose` `-V`	Increase the verboseness of fbquery functions.
`--quiet` `-Q`	Disable the verboseness of FastBit functions.

Further Details and Query Examples

The **SELECT** clause contains a list of column names separated by comma and any of the four one-argument functions such as `MIN()`, `MAX()`, `SUM()` and `AVG()`.

Each of the four functions can take only one column name as its argument and blank spaces are not allowed between the brackets and the name of the column itself. The syntax of the aggregate functions is as follows:

```
MAX(<column name>)
MIN(<column name>)
SUM(<column name>)
AVG(<column name>)
```

To retrieve only the distinct rows from the result set can be specified the flag "`-D`". If the SELECT clause is omitted, it is assumed to be "`SELECT COUNT(*)`", and the query purpose is to count the number of hits that match the conditions specified.

Suppose we have a FastBit partition simply called "`database`", some examples of query with different SELECT clause are listed below.

Example of SELECT clause with only one name of column:

```
fbquery -c "IPV4_SRC_ADDR" -d "database" -q "L4_SRC_PORT=80"
```

Example of SELECT clause with two column names separated by comma:

```
fbquery -c "IPV4_SRC_ADDR,IPV4_DST_ADDR" -d "database" -q "L4_SRC_PORT=80"
```

Examples of SELECT clause with aggregate operations and columns names:

```
fbquery -c "MAX(L4_SRC_PORT)" -d "database" -q "IPV4_SRC_ADDR=12.120.60.90"
fbquery -c "MIN(L4_DST_PORT)" -d "database" -q "L4_SRC_PORT=80"
fbquery -c "IPV4_SRC_ADDR,SUM(OUT_BYTES)" -d "database" -q "L4_SRC_PORT=80"
```

Example of SELECT with COUNT(*):

```
fbquery -c “IPV4_SRC_ADDR,COUNT(*)" -d "database" -q "L4_SRC_PORT>=0"
```

Example of the output with no SELECT clause (it counts the number of hits):

```
fbquery -d "database/" -q "L4_SRC_PORT=80"
[fbquery] Query produced 226 hits, took 0.004 CPU seconds, 0.032 elapsed seconds
```

The **FROM** clause is mandatory and it specifies the absolute or the relative pathname of a directory containing a Fastbit partition on which execute the query. Multiple directories can be defined using multiple parameters "`-d`".

Suppose we have two FastBit partitions simply called "`database1`" and "`database2`". An example of query with the clause FROM using multiple directories is shown below:

```
fbquery -c "IPV4_SRC_ADDR" -d "database1" -d "database2" -q "L4_SRC_PORT=80"
```

The **WHERE** clause is used to extract only those records that fulfill the specified criteria. It specifies the conditions of the query with a set of range conditions joined together with logical operators: `AND`, `OR`, `XOR`, and `NOT`.

The supported range conditions can be subdivided into:

- *equality conditions*
  An equality condition is defined by the equal operator "`=`" and its two operands can be arithmetic expressions, column names, numbers or string literals.

- *discrete ranges*
  A discrete range is defined by the operator "`IN`". Its syntax is: "`<column_name> IN (A,B,..,Z)`" where `A`, `B`, and `Z` can be a list of strings or numbers, comma separated. Please note that the statement "`A IN (B,C)`" is equivalent to "`A = B OR A = C`".

- *one-sided and two-sided range conditions*.
  A range condition can be one-sided as "`A > 10`", or two-sided as "`10 <= A < 20`." It can be defined with operators selected from: `<`, `<=`, `>`, and `>=`. The operands of the operator can be any arithmetic expression, column names or numbers. Alternatively, a two-side range can be defined with operators "`BETWEEN-AND`", where: "`A BETWEEN B AND C`" is equivalent to "`B <= A <= C`". The operands `A`, `B`, and `C` can be any arithmetic expression, column name or number.

Please note that an arithmetic expression may contain operators `+`, `-`, `*`, `/`, `%`, and `^` (the operator `^` denote the exponential operation). All one-argument and two-argument arithmetic functions available in the file `math.h` are supported.

Suppose we have a FastBit partition called "`database`", some examples of query with different WHERE clause are listed below.

Example of WHERE clause with an equality condition:

```
fbquery -c "L4_SRC_PORT" -d "database" -q "L4_SRC_PORT=80"
```

Example of WHERE clause with a logical operator:

```
fbquery -c "IPV4_SRC_ADDR" -d "database" -q "NOT L4_SRC_PORT=22"
```

Example of WHERE clause with the operator `IN`:

```
fbquery -c "L4_DST_PORT" -d "database/" -q "L4_SRC_PORT IN (22,25,80)"
```

Example of WHERE clause with operator `BETWEEN-AND`:

```
fbquery -c "L4_DST_PORT" -d "database" -q "L4_SRC_PORT BETWEEN 25 AND 80"
```

How to Format Query Results

The query result is printed in a CSV format in which the default delimiter is a comma. To specify a different separator can be used the parameter `"--separator"`.

Before displaying the rows that match the conditions of the query, `fbquery` displays a line containing the names of selected columns. To hide this header line can be specified the flag `"--hide-header"`.

To suitably format the results, like an IPv4 formatted in dot-notation, can be used the parameter "`--metadata-file`", followed by the pathname of the file with flow metadata dumped by nProbe. This file can be obtained from nProbe with its parameter "`--dump-metadata <file>`" and it can be customized to indicate the format in which each column must be printed.

`fbquery` formats by default only the columns that contain IP addresses in order to print their values in dotted notation (please note that the tool is able to recognize only the column names `IPV4_SRC_ADDR` and `IPV4_DST_ADDR`). To disable this automatic formatting of columns can be specified the flag "`--native-format`".

To sort in ascending order the values of specific columns can be used the parameter "`--orderby`" followed by the list of column names that want to order. Please note that the flag "`--reverse`" sorts the values in descending order in according to columns specified in the ORDER BY clause (or in the SELECT clause if no ORDER BY clause is specified).

To limit query results to those that fall within a specified range can be used the parameter "`--limit <i,j>`". The value of `i` is the starting point (remember that the first record is 0) and `j` is the duration (how many records to display). This parameter can also be used only to show the first `n` results using only one positive number as value of `i`, omitting `j`.

Some examples of query with different flags to format the results are listed below.

Examples of query with parameter `--limit` used to show only the first 5 results.

```
fbquery -c "L4_DST_PORT" -d "database" -q "L4_DST_PORT>0" --limit 5
L4_DST_PORT
1
2
3
4
5
```

How to Play with IP Addresses

As the FastBit library was not designed for handling network flows, have been implemented some missing features. On this way, for example, IP addresses can be managed in a better way.

FastBit partitions can be queried by means of `fbquery` by specifying an IP address in dotted decimal notation or with IP address range, using a subnet mask notation.

Example of query with an IPv4 in dotted decimal notation:

```
fbquery -c "IPV4_DST_ADDR,L4_DST_PORT" -d "database" -q "IPV4_DST_ADDR=10.0.0.1"
IPV4_DST_ADDR,L4_DST_PORT
10.0.0.1,80
10.0.0.1,9105
10.0.0.1,9106
```

Examples of query with IPv4 addresses with specified subnet mask:

```
fbquery -c "IPV4_DST_ADDR" -d "database" -q "IPV4_DST_ADDR=10.0.0.1/24"
IPV4_DST_ADDR
10.0.0.1
10.0.0.2
10.0.0.3
10.0.0.4
10.0.0.6
10.0.0.7
10.0.0.9
10.0.0.10
10.0.0.64
10.0.0.114
...
```

```
fbquery -c "IPV4_DST_ADDR" -d "database" -q "IPV4_DST_ADDR != 10.0.0.1/24"
IPV4_DST_ADDR
0.0.0.0
0.0.24.63
0.0.24.89
0.1.0.6
0.10.149.85
0.18.218.179
0.21.92.100
0.21.216.230
0.21.217.44
...
```

Appendix A
NetFlow v9/IPFIX Flow Format

The `-T` flag (as `--fastbit-template` and `--fastbit-index`) enabled users to specify the format of NetFlow v9/IPFIX flows. The format options currently supported by nProbe are those specified in the NetFlow v9 RFC, namely (in square brackets it is specified the field Id as defined in the RFC):

```
[1]    %IN_BYTES                 Incoming flow bytes
[2]    %IN_PKTS                  Incoming flow packets
[3]    %FLOWS                    Number of flows
[4]    %PROTOCOL                 IP protocol byte
[132]  %PROTOCOL_MAP             IP protocol name
[5]    %SRC_TOS                  Type of service byte
[6]    %TCP_FLAGS                Cumulative of all flow TCP flags
[7]    %L4_SRC_PORT              IPv4 source port
[135]  %L4_SRC_PORT_MAP          IPv4 source port symbolic name
[8]    %IPV4_SRC_ADDR            IPv4 source address
[9]    %SRC_MASK                 Source subnet mask (/<bits>)
[10]   %INPUT_SNMP               Input interface SNMP idx
[11]   %L4_DST_PORT              IPv4 destination port
[139]  %L4_DST_PORT_MAP          IPv4 destination port symbolic name
[12]   %IPV4_DST_ADDR            IPv4 destination address
[13]   %DST_MASK                 Dest subnet mask (/<bits>)
[14]   %OUTPUT_SNMP              Output interface SNMP idx
[15]   %IPV4_NEXT_HOP            IPv4 next hop address
[16]   %SRC_AS                   Source BGP AS
[17]   %DST_AS                   Destination BGP AS
[21]   %LAST_SWITCHED            SysUptime (msec) of the last flow pkt
[22]   %FIRST_SWITCHED           SysUptime (msec) of the first flow pkt
[23]   %OUT_BYTES                Outgoing flow bytes
[24]   %OUT_PKTS                 Outgoing flow packets
[27]   %IPV6_SRC_ADDR            IPv6 source address
[28]   %IPV6_DST_ADDR            IPv6 destination address
[29]   %IPV6_SRC_MASK            IPv4 source mask
[30]   %IPV6_DST_MASK            IPv4 destination mask
[32]   %ICMP_TYPEICMP            Type * 256 + ICMP code
[34]   %SAMPLING_INTERVAL        Sampling rate
[35]   %SAMPLING_ALGORITHM       Sampling type (deterministic/random)
[36]   %FLOW_ACTIVE_TIMEOUT      Activity timeout of flow cache entries
[37]   %FLOW_INACTIVE_TIMEOUT    Inactivity timeout of flow cache entries
[38]   %ENGINE_TYPE              Flow switching engine
[39]   %ENGINE_ID                Id of the flow switching engine
[40]   %TOTAL_BYTES_EXP          Total bytes exported
[41]   %TOTAL_PKTS_EXP           Total flow packets exported
[42]   %TOTAL_FLOWS_EXP          Total number of exported flows
[56]   %IN_SRC_MAC               Source MAC Address
[57]   %OUT_DST_MAC              Destination MAC Address
[58]   %SRC_VLAN                 Source VLAN
[59]   %DST_VLAN                 Destination VLAN
[60]   %IP_PROTOCOL_VERSION      [4=IPv4][6=IPv6]
[61]   %DIRECTION                [0=ingress][1=egress] flow
[70]   %MPLS_LABEL_1             MPLS label at position 1
[71]   %MPLS_LABEL_2             MPLS label at position 2
[72]   %MPLS_LABEL_3             MPLS label at position 3
[73]   %MPLS_LABEL_4             MPLS label at position 4
[74]   %MPLS_LABEL_5             MPLS label at position 5
[75]   %MPLS_LABEL_6             MPLS label at position 6
[76]   %MPLS_LABEL_7             MPLS label at position 7
```

```
[77]   %MPLS_LABEL_8             MPLS label at position 8
[78]   %MPLS_LABEL_9             MPLS label at position 9
[79]   %MPLS_LABEL_10            MPLS label at position 10
[80]   %FRAGMENTED               1=some flow packets are fragmented
[81]   %FINGERPRINTTCP           fingerprint
[82]   %CLIENT_NW_DELAY_SEC      Network latency client <-> nprobe (sec)
[83]   %CLIENT_NW_DELAY_USEC     Network latency client <-> nprobe (usec)
[84]   %SERVER_NW_DELAY_SEC      Network latency nprobe <-> server (sec)
[85]   %SERVER_NW_DELAY_USEC     Network latency nprobe <-> server (usec)
[86]   %APPL_LATENCY_SEC         Application latency (sec)
[87]   %APPL_LATENCY_USEC        Application latency (sec)
[96]   %IN_PAYLOAD               Initial payload bytes
[97]   %OUT_PAYLOAD              Initial payload bytes
[98]   %ICMP_FLAGS               Cumulative of all flow ICMP types
```

Plugin SIP templates:

```
[130]  %SIP_CALL_IDS             IP call-id
[131]  %SIP_CALLING_PARTY        SIP Call initiator
[132]  %SIP_CALLED_PARTY         SIP Called party
[133]  %SIP_RTP_CODECS           SIP RTP codecs
[134]  %SIP_INVITE_TIME          SIP SysUptime (msec) of INVITE
[135]  %SIP_TRYING_TIME          SIP SysUptime (msec) of Trying
[136]  %SIP_RINGING_TIME         SIP SysUptime (msec) of RINGING
[137]  %SIP_OK_TIME              SIP SysUptime (msec) of OK
[138]  %SIP_ACK_TIME             SIP SysUptime (msec) of ACK
[139]  %SIP_RTP_SRC_PORT         SIP RTP stream source port
[140]  %SIP_RTP_DST_PORT         SIP RTP stream dest port
```

Plugin Efficiency calculation templates

```
[165]  %EFFICIENCY_SENT          Avg. transmission efficiency % (send)
[166]  %EFFICIENCY_RCVD          Avg. transmission efficiency % (rcvd)
```

Plugin Video protocol detection (skeleton plugin) templates:

```
[188]  %VIDEO_PROTO              Simple counter
```

Plugin SMTP Protocol Dissector templates

```
[185]  %SMTP_MAIL_FROM           Mail sender
[186]  %SMTP_RCPT_TO             Mail recipient
```

Plugin Flow Serial Identifier templates

```
[190]  %FLOW_ID                  Serial Flow Identifier
```

Plugin HTTP Protocol Dissector templates

```
[180]  %HTTP_URL                 HTTP URL
[181]  %HTTP_RET_CODE            HTTP return code (e.g. 200, 304...)
```

Plugin dump templates

```
[100]  %DUMP_PATH                Path where dumps will be saved
```

Plugin RTP templates

```
[150]  %RTP_FIRST_SSRC           First flow RTP Sync Source ID
[151]  %RTP_FIRST_TS             First flow RTP timestamp
[152]  %RTP_LAST_SSRC            Last flow RTP Sync Source ID
[153]  %RTP_LAST_TS              Last flow RTP timestamp
[154]  %RTP_IN_JITTER            RTP Jitter (ms * 1000)
[155]  %RTP_OUT_JITTER           RTP Jitter (ms * 1000)
[156]  %RTP_IN_PKT_LOST          Packet lost in stream
[157]  %RTP_OUT_PKT_LOST         Packet lost in stream
[158]  %RTP_OUT_PAYLOAD_TYPE     RTP payload type
[159]  %RTP_IN_MAX_DELTA         Max delta between consecutive pkts
[160]  %RTP_OUT_MAX_DELTA        Max delta between consecutive pkts
```

Example

If you want to specify NetFlow v9 flows in a format similar to NetFlow v5 flows, you can do as follows:

```
nprobe -T "%IPV4_SRC_ADDR %IPV4_DST_ADDR %IPV4_NEXT_HOP %INPUT_SNMP
%OUTPUT_SNMP %IN_PKTS %IN_BYTES %FIRST_SWITCHED %LAST_SWITCHED
%L4_SRC_PORT %L4_DST_PORT %TCP_FLAGS %PROTOCOL %SRC_TOS %SRC_AS
%DST_AS %SRC_MASK %DST_MASK"
```

Note that the fields start with a `%` symbol and they are separated by a space.