An open source network traffic performance monitoring and diagnostics tool.

www.trafmon.org

Configuration & Administration Guide

Thomas Grootaers, Luc Lechien

Software Release 1.0

2020-10

COPYRIGHT, LICENSE AND TRADEMARKS

This material is based upon work funded and supported by the European Space Agency and the Belgian Federal Authorities (BELSPO) under GSTP Contract Nr ESRIN 4000128964/19/I-EF with AETHIS sa/nv, Belgium.

DOCUMENT HISTORY

Release	Date	Change
1.0	Sept 2020	First issue

ACKNOWLEDGEMENTS

The authors wish to acknowledge the valuable contributions of all ancient employees of the AETHIS® Company in Belgium, who have worked on the successive versions of the base software and its documentation from which the open source trafMon software is derived.

In particular, special recognition is given to Jacques Maes, David Orban, Jonathan Van den Schrieck, Benoît Liétaer, Julien Denis, Thomas Soupart, Fabien Coenegrachts, who have more specifically participated to its elaboration. Also a thought is given in memory the authors' deceased associate, Luc Steenput, who has heavily promoted the initial idea and subsequent enhancements of the tool, within the European Space Agency and elsewhere.

Lastly, the authors wish to acknowledge the strong support of ESA staff members: Manfred Lugert, Erling Kristiansen, Johan Stjernevi, Manfred Bertelsmeier, Gioacchino Buscemi, Michele Iapaolo, Andrea Cogliandro and Claudia Neroni, as well as of officers of the Belgian BELSPO Federal Service, Jacques Nijskens, Agnès Grandjean and Hendrick Verbeelen.

TABLE OF CONTENT

ACRONYMS AND ABBREVIATIONS

1. TRAFMON COMPUTING ENVIRONMENT

1.1 TRAFMON MACHINES ARCHITECTURE

The trafMon tool deployment typically involves two types of components:

- The traffic probe machine and
- The central processing and/or database server.

1.1.1 The Traffic Probe Machine

The traffic probe is typically a physical (micro) computer, dedicated to this task, and that the user would replicate in the different locations/sites where traffic of interest is to be monitored.

The use of virtual machine is discouraged for this, because of the necessary accuracy in network packet sampling timestamp and the performance required to absorb the potential traffic peaks.

The probe runtime software consists in a pipeline of two permanent processes linked by an in-memory shared circular buffer. The father process performs the actual in situ protocol decoding by inspecting the successive packets (Ethernet frames) directly in the kernel-resident buffer and transmitting its filtered decoding results to its child. The child process is in charge of the stateful analysis of protocol exchanges, of the accumulation of the traffic observations statistics and of the continuous delivery of the resulting observations to one (or more) central collector(s).

Multi-core CPU (or multi-CPU) architecture is exploited by letting the father process locks itself on one core and the child assigning itself to the remaining ones.

The size of the RAM is capital to create the largest possible in-memory kernel packet capture ring buffer and the shared memory inter-process queue.

One probe can have several probing ports: typically dedicated Ethernet interfaces (whose on-chip packet reassembly and upper protocol pre-processing is voluntarily de-activated), without being assigned an IP address.

Practical tests, conducted in 2016 on modern computer hardware, through the replay at full speed of pre-captured traffic dumps, have demonstrated that the probe can afford fully filled (1500 bytes) packets data rate at 2.3 Gbps. Thus, when mirroring a Fast Ethernet switch traffic, the best is to mirror to a 10 Gbps interface, to avoid the switch dropping mirror packets in excess, and to dimension the probe kernel capture buffer in such a way as to cope with high rate burst of traffic.

In current version, passive taps, where packets in each direction are mirrored to 2 separate Ethernet ports, are not supported: minor changes are required in the software to respect the order of occurrence in stateful analysis of packets exchanged.

The probe computer performances is thus determined by the multi-CPU speed, the RAM speed and the amount of RAM. One (or more) 10 Gbps native Ethernet port is preferred as dedicated traffic capture interface.

The remote administration and the probe-to-collector UDP-based data flow can be ideally conducted via one (or two) separate Fast Ethernet port(s), because data rate is less an issue. USB-to-Ethernet adapter(s) can even be used for circumventing base computer limitations.

1.1.2 The Central Server Machine

The central trafMon collector process is relatively lightweight. Its role is mostly to dump the raw observations received by the disseminated probes into ASCII log files. It is a bit more involved in buffering and conciliating individual packet observations from different probes to detect packet losses or to compute packet one-way latencies (and jitter – to be done).

When NetFlow/sFlow/IPFIX add-on is also needed, the CERT® SiLK rwflowpack daemon can co-exist on the collector computer. It also simply dumps the flow records packets into ASCII logs.

So, the central raw data collection is only demanding in sufficient disk capacity for the temporary logs, but more significantly for storing archives of raw data logs.

When NetFlow/sFlow/IPFIX data source is also used, the CERT® SiLK rwfilter command is involved in extracting the collected records. Unfortunately, its **finest granularity is one hour**! So every hour, a rather big NetFlow observations file is produced. And its per-flow records can lapse over several minutes. So a local pre-processing of this big log file is required to the volume of each reported dataflow during every minute of its duration. The time taken by the processing can form a non-negligible part of the overall lapse of a regular observations pre-processing batch; hence influencing the highest affordable frequency at which this database refresh can be conducted (hopefully every 10 minutes). In that case of use this NetFlow observations preliminary treatment reveals dimensioning for required computing performance of the data collection function.

What is quite heavier is the regular process of raw observations pre-processing, and database loading and update, as well as the resources available to the Tomcat server drawing the on-demand reports.

This is influenced by the performance of the system running the MySQL database server: RAM type (DDR4), disk speed (Solid State), amount of RAM (I/O cache size, `mysqld` database server and tomcat `java` resident set size), CPU rate running the single-threaded database server

When the database server is running on a separate computer, the performance of the regular batch processing is mostly impacted by the database server disk I/O rate (e.g. multiple high-performance SSD devices configured in hardware striping: RAID 0) and RAM size and speed. And the affordable history length of the per-minute (and per-hour) detailed performance figures is determined by the database files disk partition capacity (partition of more than one 2 TB implies the use of UEFI and GPT in booting and disk partitioning).

The trafMon runtime database is solicited by three different types of requests:

- The regular database loading and aggregation mostly consists in bulk loading (LOAD DATA LOCAL INFILE) into TEMPORARY tables, then updating the persistent tables, through dedicated SQL stored procedures, at 1 minute, 1 hour and 1 day granularity with aggregated fresh data.
- The daily update of volume related data tables feeding the synthetic views of traffic in terms of Activity/Location/Host and peer Countries is a rather heavy processing, conducted by recurrent stored procedures.
- The on-the-fly report generation, where the menu bar dynamic contents is updated by PHP online queries. The generation of details reports does simply retrieve the relevant records out of prepared database tables. But the generation of synthesis reports rely on a potentially heavy stored procedure, which retrieves data from daily aggregated volume tables, but further aggregate them based on the dynamically selected report parameters.

The regular loading typically induces a peak load every 10 minutes (all the time) and a longer load, once per hour, for the NetFlow data.

The daily work, also encompassing the more seldom database clean-up maintenance must be carefully spread over the night.

The on-the-fly report generation cannot be controlled, but typically happens during the day. It is the less predictable while the more demanding database activity, imposing short response time due to its interactive nature.

In summary, the operational database server must be chosen as best-on-market, with carefully tuned hardware components and operating system configuration.

The Web interface and report generation function can theoretically run together with the collector function and maybe the database service. But it induces peak of Tomcat/Java processing of the database retrieved data upon formatting of a requested report. The

Tomcat process consumes a lot of RAM and competes therefore with the MySQL server and the disk I/O buffers.

Note that nothing prevents that those central functions are assigned to one or more virtual server, provided that the reporting has enough CPU and RAM and that the database service has high data rate disk I/O (SSD, hardware RAID 1 and sizeable high-speed RAM buffers) and a lot of high-speed RAM also available to the MySQL server.

1.2 OPERATING SYSTEMS

The specific trafMon software has been developed on Linux, more specifically tested on an old Debian (x86 32 bit) and operated on not fully up-to-date CentOS 7.2 and 7.3 releases (x86_64).

It is for sure that the trafMon probe (***tmon_probe***) and trafMon collector (***tmon_collector***) will be easily compiled on any vanillas of Linux, and even of BSD and other *NIX versions. But the probe has specific performance enhancements only available on Linux (packet capture ring buffer, deactivation of Ethernet card specific TCP/IP protocol handling, process locking on specific CPU core).

In a first time, the pre-compiled **binary distribution packages** will be available for **CentOS Linux release 7.3.1611 (Core), Linux kernel 3.10.0-514.26.1.el7.x86_64**. This is our baseline for the whole software installation. Nevertheless, the Angular.js development environment for the JavaScript menu bar is built on Microsoft Windows™ and the Eclipse BIRT Designer, for editing the trafMon report templates, has been indifferently run on the CentOS and on MS Windows systems.

For add-on support of NetFlow/sFlow/IPFIX, two binaries of the CERT® SiLK open source distribution are required: **rwflowpack** and **rwfilter**. Although we were running them on the same CentOS baseline server platform, it is perfectly possible to have them installed on other supported operating systems. The link with the trafMon processing chain is that the processed output of the **rwfilter** is saved, every hour, as a file in the raw observation spool directory together with the output log files produced by the **tmon_collector**.

The MySQL database can be installed and run on any platform supported by the MySQL 5.6 release. This can be commercial or open source distribution of Linux of other UNIX™ variants (e.g. Solaris™) or even a Microsoft Windows platform.

Our installation of the BIRT Report Engine runtime is using Apache Tomcat 7, installed on our central CentOS 7.3 server. But it can run in other J2EE application servers instead (IBM

WebSphere®, Oracle WebLogic®, Red Hat JBoss®), among others the OpenText BIRT iHub™ commercial server (OpenText® bought Actuate® Corporation, the company that has developed BIRT). So you have the choice of the Java server component and therefore on its underlying operating system platform.

2. INSTALLATION CONVENTIONAL BASELINE

The following conventions can be modified by the trafMon tool administrator, but they influence default values of arguments/parameters of scripts and sample configuration files and forms the baseline for examples provided in the trafMon documentation.

It is assumed that, at the exception of `logrotate` and (re-)start of MySQL, Tomcat and, optionally SiLK `rwflowpack`, all tasks related to trafMon tool execution is performed by a common Linux account named `trafmon` with primary group `trafmon`. He is the owner of every configuration file and of the data and message log files produced by the tool execution.

NOTE:

> The trafMon Software Installation Guide explains how to run the packet capture `tmon_probe` without root privileges.

By convention, the trafMon own binaries and executable scripts are located in `/opt/trafMon/bin`. This can be achieved by extracting the production ready distribution under `/opt` and by renaming (or linking) the root directory `trafMon-x.y.z…` as `trafMon/.`

By convention the BIRT Runtime provided scripts “genreport.sh” is linked as `/opt/trafMon/bin/genreport.sh`.

By convention, the trafMon tool own run-time configuration files are placed on a structure rooted at `/etc/trafMon/`:

- `/etc/trafMon/xml/tmon.dtd` is the run-time configuration definition file referred to by:
- `/etc/trafMon/xml/tmon.xml`, the default run-time configuration file common to every trafMon probe and to the trafMon collector(s);
- `/etc/trafMon/xml/tmon-new.xml` is the default name of a new common version of the configuration file (ideally with a **`startAt=`** in the future), forcing all probes and collector(s) to auto-restart (simultaneously) with the update config.

- `/etc/trafMon/diag/`*`myprobe1`*`.diag` is the run-time configuration for the error and diagnostic tracing messages logging for the specific instance of `tmon_probe` named *myprobe1*, and
- `/etc/trafMon/diag/`*`mycollector`*`.diag` is the run-time configuration for the error and diagnostic tracing messages logging for the specific instance of `tmon_collector` named *mycollector*,

- `/etc/trafMon/cred/db.cred` is an owner-read-only file specifying the database server URL and run-time trafMon database name, as well as two dedicated MySQL accounts (name and rot13 encoded password):
  - tmon_db is the default user for the creation, loading in and modifying the trafMon database(s),
  - tmon_birt is the default user for the read-only data retrieval in producing the trafMon reports and for the batch execution of procedure stored in the database `trafMom_template`.

- `/etc/trafMon/report/`*`mydetails`*`.genAddrs` is an optional file with IPv4 address patterns (regular expressions) to determine which data flow instances are subject to (optional) automatic generation of the set of protocol details as PDF documents. Produced documents are in a hierarchy rooted at `/var/trafMon/reports/`*`mydetails`*`.`

- `/etc/trafMon/ipInfo.ini` is the list of known IPv4 addresses and subnets that are classified in terms of Activities and Locations. *The most complete and up-to-date is this file, the more accurate and useful will be the trafMon top-level synthesis reports*!

When the add-on NetFlow optional data source is used (thanks to the open source CERT® SiLK package), the software is assumed to be installed under `/usr/local` base directory, hence the relevant executables are `/usr/local/bin/rwfilter` and `/usr/local/bin/rwcut`. The service is supposed to be registered by `/etc/init.d/rwflowpack`.

The main configuration file is in `/usr/local/etc/rwflowpack.conf`. The (maybe original) silk.conf(8) file is `/var/silk/twoway-silk.conf` and the custom definitions of "*SiLK probe sensors*" – the senders of NetFlow/sFlow/IPFIX records packets – is given by `/var/silk/sensor.conf`. The SiLK data log files hierarchy is rooted at `/var/silk/data/`.

The script trafMon_FormatNetFlow.py is supposed to gather the accumulated NetFlow records every hour and to produce the file `/var/trafMon/collector/`*`YYY-MM-DD`*`Thh:00`, mixed with the `tmon_collector` output data logs.

The MaxMind® GeoIP™ GeoLite2™ database files (or their more accurate commercial versions) are assumed to be located as `/var/trafMon/GeoIP/GeoIPASNum.dat` and `/var/trafMon/GeoIP/GeoLite2-City.mmdb`.

The `tmon_collector` output data log files accumulate under `/var/trafMon/collector/`:

- `observations.`*`YYYMMDD`*`T`*`hhmm`*`.flow`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.metrc`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.hops`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.1wobs`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.1wlost`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.1wct`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.latcy`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.ipct`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.ipsz`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.icmpct`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.udpct`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.tcpct`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.tcpcon`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.ftpct`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.ftpxfr`,
  `observations.`*`YYYMMDD`*`T`*`hhmm`*`.2way`
- `events.`*`YYY-MM-DD`*`T`*`hhmm`*`.evt`
- `exceptions.`*`YYY-MM-DD`*`T`*`hhmm`*`.1wmiss`,
  `exceptions.`*`YYY-MM-DD`*`T`*`hhmm`*`.1wdrop`

During loading regular processing, these observations are moved under `/var/trafMon/collector/work/` then these are merged in one file per observation type, under `/var/trafMon/collector/mergedFiles/` and the files are archived in a compressed tarfile `/var/trafMon/collector/done/`*`YYYMMDD`*`T`*`hhmm`*`.tbz`. Finally, the files are put under `/var/trafMon/collector/toLoad/`*`suffix`*, then removed after successful database loading and aggregation (*suffix* is the short name of the observation type, as above).

The processing trace log files of all component of the trafMon system are assumed to be produced under /var/log/trafMon/*xyz*.log. These would be owned by the user:group `trafmon:trafmon`. These are “logrotated” every hour of upon size reaching 200MB.

3. SCRIPTS AND BINARIES REFERENCE GUIDE

3.1 TRAFMON PROBE

```
$ ./tmon_probe
USAGE:
tmon_probe [-l] [-c configXML] [-n NEWconfigXML] probeName
        -l (local) means using ./tmon.xml, ./tmon-new.xml and,
                        if it exists, ./tmon_probe.diag
        -c means use the given XML, and nothing else
        -n monitors the given NEW XML for scheduled config update
                        (based on its `startAt' attribute
 If -l is NOT given, /etc/trafMon/xml/tmon.xml is used
 If -l and -n are NOT given, /etc/trafMon/xml/tmon-new.xml is looked at
 If -l is NOT given, or ./tmon_probe.diag doesn't exist,
                        /etc/trafMon/diag/<probeName>.diag is used
```

The XML configuration file contains the configuration of all the probe(s) and collector(s). In this file, the probe/collector selects the parts relative to it and ignores the others. As such, the same XML configuration file can be used for each probe/collector.

When detecting the presence of /etc/trafMon/xml/tmon-new.xml, when its “startAt=” value is past, the probe restarts immediately on this new config, renamed tmon.xml. Otherwise it does it when indicated.

On the other hand, the “.diag” file is used to configure the verbosity of the logs.

IMPORTANT

The probe requires packet capture privileges. Either it must be launched under the `root` account, but then it creates its diagnostic log file(s) as owned by `root`; or the binary (`/opt/tmon/bin/tmon-probe`) is assigned specific capabilities for the executable file owner:

```
root # setcap cap_net_raw,cap_net_admin=eip /opt/tmon/bin/tmon_probe
root # chown trafmon:trafmon /opt/tmon/bin/tmon_probe
```

In such case, the program can be launched by unprivileged account member of the `trafmon` group. However, due to the added capabilities, the program does not generate a core file upon abortion, nor can it be killed by unprivileged user.

Note that the probe command gives rise to two processes. The father appears with its normal command-line, while the child has its name partly re-written in the process list.

```
trafmon@trafMon-prb$ ps aux | fgrep tmon_probe | fgrep –v grep
trafmon   7402  9.5 17.1 564044 455356 ?       S    Sep22 3864:00
tmon_probe -c /etc/tmon/xml/singleProbe.xml trafMon-prb
trafmon   7403  8.3  0.9 238608 25008 ?        S    Sep22 3359:34
tmon_probe(Child) /tmon/xml/factseo2_for_FAT.xml trafMon-prb
```

In production, the probe is typically started (and automatically re-started) through crontab scheduling of the `trafMon_probeResurrect.sh` script (see section 3.8 below).

For tests, or manually controlled environment, you can start the (father) probe in foreground, in the persistent shell console provided by the **third-party `screen` utility**:

The probe system can be started on the system **trafMon-prb** as root.

```
root@trafMon-prb# screen
root@trafMon-prb# cd /opt/tmon/bin
root@trafMon-prb# ./tmon_probe -c /etc/tmon/xml/myconfig.xml trafMon-prb
root@trafMon-prb# <ctrl>a d
```

The <ctrl>a d command will disconnect the screen from the terminal. Then you may disconnect from the trafMon-prb2 host.

To reconnect to the screen session terminal (no further argument when it's the only screen session pending):

```
root@trafMon-prb# screen -r
```

3.2 TRAFMON COLLECTOR

```
$ ./tmon_collector
USAGE:
tmon_collector [-l] [-c configXML] [-n NEWconfigXML] collectorName
        -l (local) means using ./tmon.xml, ./tmon-new.xml and, if it
exists, ./tmon_collector.diag
        -c means use the given XML, and nothing else
        -n monitors the given NEW XML for scheduled config update
                        (based on its `startAt' attribute
 If -l is NOT given, /etc/trafMon/xml/tmon.xml is used
 If -l and -n are NOT given, /etc/trafMon/xml/tmon-new.xml is looked at
 If -l is NOT given, or ./tmon_collector.diag doesn't exist,
                        /etc/trafMon/diag/<collectorName>.diag is used
```

The XML configuration file contains the configuration of all the probe(s) and collector(s). In this file, the probe/collector selects the parts relative to it and ignores the others. As such, the same XML configuration file can be used for each probe/collector.

When detecting the presence of /etc/trafMon/xml/tmon-new.xml, when its “startAt=” value is past, the probe restarts immediately on this new config, renamed tmon.xml. Otherwise it does it when indicated.

On the other hand, the “.diag” file is used to configure the verbosity of the logs.

In production, the probe is typically started (and automatically re-started) through crontab scheduling of the `trafMon_serverResurrect.sh` script (see section 0 below).

For tests, or manually controlled environment, you can start the (father) probe in foreground, in the persistent shell console provided by the **third-party `screen` utility**:

The probe system can be started on the central system **trafMon-svr** as trafmon.

```
trafmon@trafMon-svr % screen
trafmon@trafMon-svr % cd /opt/tmon/bin
trafmon@trafMon-svr % ./tmon_collector -c /etc/tmon/xml/myconfig.xml
collector
trafmon@trafMon-svr % <ctrl>a d
```

The <ctrl>a d command will disconnect the screen from the terminal. Then you may disconnect from the trafMon-svr host

To reconnect to the screen session terminal (no further argument when it’s the only screen session pending):

```
trafmon@trafMon-svr % screen -r
```

3.3 EXTRACTION FROM SILK NETFLOW RECORDS LOGS

```
$ ./trafMon_FormatNetFlow.py -h
Usage: trafMon_FormatNetFlow.py [options]

Options:
  -h, --help            show this help message and exit
  -H nbOfHours, --hour=nbOfHours
                        Number of hours of data to process (starting from the
                        past hour). This option has precedence to the
                        start/end date options. E.g.: '-H 2' at 14:36:27 means
                        data for [12:00:00, 14:00:00]
  -S startTime, --dateStart=startTime
                        Time at which the collection of data will begin
                        (format: YYYY/MM/DD:HH). The data for the given hour
                        are included in the results.
  -E endTime, --dateEnd=endTime
                        Time at which the collection of data will end (format:
                        YYYY/MM/DD:HH). The data for the given hour are
                        included in the results.
  -N SRCPATH, --netflowdata=SRCPATH
                        Path to the root of the SiLK collected NetFlow data
                        (default: /var/silk/data/)
  -D DSTPATH, --collectordata=DSTPATH
                        Path to the runtime observations logs produced by the
                        trafMon Collector, where to place the NetFlow
                        extracted log (default: /var/trafMon/collector/)
  -C SILKCONFIG, --silkconfig=SILKCONFIG
                        Path to SiLK config file (default: /var/silk/twoway-
                        silk.conf)
  -s SILKBIN, --silkbin=SILKBIN
                        Path where to find SiLK rwfilter and rwcut utilities
                        (default: /usr/local/bin/)
  -L LOGDIR, --logFileDirectory=LOGDIR
                        Path to log directory (default: /var/log/trafMon/)
```

This script uses `rwfilter` and `rwcut` SiLK utilities and a big pipeline of shell formatting commands to extract and format a log file similar to those produced by the `tmon-collector`.

Typically, the –H option is used in the cron job to create a new data file every hour. On the other hand, the start/end time options are used when data for a specific range of time is needed.

Note that a file is created for every hour of data, with the following format: “netflowYYYY-MM-DDTHH:00.netfl” (e.g.: netflow2016-09-19T14:00.netfl), where YYYY-MM-DDTHH is the time/hour during which all the records contained in the file started (a record can end in the next hour though).

Also note that if no data is available for a given hour, then the file created for this hour is automatically removed at the end of the execution.

This script also uses the "`twoway-silk.conf`" configuration file, which should be located in */var/silk/twoway-silk.conf*.

3.4 DATABASE LOADER

```
$ ./trafMon_loader.py -h
Usage: trafMon_loader.py [options]

Options:
  -h, --help            show this help message and exit
  -p PATH, --path=PATH  Path to the directory containing the data file to load
                        into the DB (default: /var/trafMon/collector/
  -l, --localConfig     If -l is specified, the db.cred file in the current
                        directory is used. Otherwise, the default
                        /etc/tmon/cred/db.cred file is loaded
  -L LOGFILEDIRECTORY, --logFileDirectory=LOGFILEDIRECTORY
                        Path to log directory (default: /var/trafMon/log/)
```

This script is typically scheduled every 10 minutes, although the optional NetFlow data log cannot be produced more frequently than once per hour.

trafMon collector observations are first moved under `/var/trafMon/collector/work/` then these are merged in one file per observation type, under `/var/trafMon/collector/mergedFiles/` and the files are archived in a compressed tarfile `/var/trafMon/collector/done/`*YYYMMDD*`T`*hhmm*`.tbz`. Finally, the files are put under `/var/trafMon/collector/toLoad/`*suffix*, then removed after successful database loading and aggregation. *Suffix* is the short name of the observation type. The corresponding radix of database table names is:

```
    "flow":"flowtable",
    "ipct":"ipcttable",
    "ipsz":"ipsztable",
    "icmpct":"icmpcttable",
    "udpct":"udpcttable",
    "tcpct":"tcpcttable",
    "ftpct":"ftpcttable",
    "tcpcon":"tcpcontable",
    "ftpxfr":"ftpxfrtable",
    "metrc":"metrictable",
    "2way":"twowaydelaytable",
    "latcy":"onewaylatencytable",
    "hops":"hopstable",
    "1wct":"onewaycttable",
    "1wobs":"onewaydelaytable",
    "1wmiss":"onewaymisstable",
```

```
    "1wlost":"onewaylosttable",
    "evt":"eventtable",
    "netfl":"netflowtable"
```

For NetFlow, a rather heavy processing is performed first: each flow that spans more than one minute has its volumes and packets values equally distributed over every crossed minutes, so that the data are compatible with the ipct counters produced by trafMon.

When the loader crashes/aborts, its will merge unloaded data with new observations at next run.

3.5 PARTIAL OR FULL UPDATE OF INFORMATION ABOUT DISCOVERED IPV4 ADDRESSES

```
$ ./trafMon_updateIpInfo.py
Usage: trafMon_updateIpInfo.py [options]

Options:
  -h, --help            show this help message and exit
  -p INFOFILE, --path=INFOFILE
                        File pathname to the .ini file with information about
                        the known IP adresses/segments. (Default:
                        /etc/trafMon/ipInfo.ini)
  -l, --localConfig     If -l is specified, the db.cred file in the current
                        directory is used. Otherwise, the default
                        /etc/tmon/cred/db.cred DB configuration file is loaded
  -g GEOIPPATH, --geoipPath=GEOIPPATH
                        Path to the directory with GeoLite2 or Maxmind GeoIP2
                        databases (Default: /var/trafMon/GeoIP/)
  -a, --all             Activate a full update of the ipInfoTable rather than
                        a partial one (partial means that we do not try to
                        resolve addresses which have already been successfully
                        resolved previously).
  -L LOGDIR, --logFileDirectory=LOGDIR
                        Path to log directory (default: /var/log/trafMon/)
```

Tries to decorate the database table `ipinfotable` of discovered IPv4 addresses with either the corresponding Activity/Location (found in /etc/trafMon/ipInfo.ini), or with Country/City/ASN found in the MaxMind database files.

It also attempts to resolve their DNS name.

Partial execution is foreseen after every data loading, to update not yet DNS resolved addresses (e.g. newly discovered). Refresh for “ALL” could be done once a week or less. It gives rises to successive bursts of DNS queries.

3.6 BATCH PRODUCTION OF SYNHESIS REPORT(S)

```
$ ./trafMon_volumeReportGen.py -h
Usage: trafMon_volumeReportGen.py [options]

Options:
  -h, --help            show this help message and exit
  -d DBname, --db=DBname
                        Database to be used. Default to 'trafMon'.
  -r report, --report=report
                        Type of synthesis report to be generated. Possible
                        choices are: [manager, operator, conversation].
                        Default to 'manager'.
  -D destination, --destination=destination
                        Destination directory. Default to
                        '/var/trafMon/reports/2020/10/01', where the
                        'YYYY/MM/DD' part is the generation time of the report
                        (today).
  -t top, --top=top     Top-N to be used. Possible choices are: [5, 10, 15,
                        20, 25]. Default to top-5.
  -T threshold, --threshold=threshold
                        Threshold bandwidth in b/s to be used. Possible
                        choices are: [0, 1000, 10000, 50000, 100000, 500000].
                        Default to 1000.
  -A activityName, --activity=activityName
                        Activity to be used. Default to 'any'. Use quotes if
                        the activity name include a space.
  -L locationName, --location=locationName
                        Location to be used. Default to 'any'. Use quotes if
                        the location name include a space.
  -H hostName, --host=hostName
                        Host to be used. Default to 'any'. Use quotes if the
                        host name include a space.
  -s startDate, --startDate=startDate
                        Start date to be used (format: 'YYYY-MM-DD'). Default
                        to first day of previous month.
  -e endDate, --endDate=endDate
                        End date to be used (format: 'YYYY-MM-DD'). Default to
                        last day of previous month.
  -l LOGDIR, --logFileDirectory=LOGDIR
                        Path to log directory (default: /var/log/trafMon/)
  -R TEMPLATESFOLDER, --reportTemplatesDirectory=TEMPLATESFOLDER
                        Path to trafMon report templates directory. Default:
                        /opt/trafMon/trafMon_reports/
  -g GENREPORT, --genReport_sh=GENREPORT
                        Full pathname to the Birt runtime 'genReport.sh'
                        utility. Default to /opt/trafMon/bin/genReport.sh,
                        which is typically a symbolic link to the BIRT RunTime
                        installation/ReportEngine/genReport.sh
```

Permits to generate a report of one of the three types of synthesis reports, when the same kind of parameters selection provided by the trafMon interactive Web menu bar application.

By default, the report is produced in a directory path based on the data of execution of this command (not necessarily related to the report span boundaries):
`/var/trafMon/reports/YYYY/MM/DD/`

3.7 BATCH PRODUCTION OF DETAILS REPORTS

```
$ ./trafMon_detailReportGen.py -h
Usage: trafMon_detailReportGen.py [options]

Options:
  -h, --help            show this help message and exit
  -l, --localConfig     if -l is specified, db.cred file is fetched from the
                        current directory. Default: /etc/trafMon/cred/
  -f FILENAME, --filename=FILENAME
                        Give a pathname or filename containing IP address
                        patterns in concerned Flow Instances. This file
                        basename is also the root of the tree of generated
                        reports. When relative, the file is fetched from
                        current directory when -l is specified, otherwise from
                        default /etc/trafMon/report/
  -D destination, --destination=destination
                        Destination directory. Default to
                        '/var/trafMon/reports/'.
  -s STARTDATE, --startDate=STARTDATE
                        Give a start date in format: 'YYYY-MM-DD'
  -e ENDDATE, --endDate=ENDDATE
                        Optionally give an end date in format: 'YYYY-MM-DD'
  -t TIMESPAN, --timespan=TIMESPAN
                        Without endDate: choose between 'weekly' or 'monthly'
                        report. With endDate: give any identifier for this
                        type of reports.
  -L LOGDIR, --logFileDirectory=LOGDIR
                        Path to log directory. Default: /var/log/trafMon/
  -T TEMPLATESFOLDER, --reportTemplatesDirectory=TEMPLATESFOLDER
                        Path to trafMon report templates directory. Default:
                        /opt/trafMon/report/
  -g GENREPORT, --genReport_sh=GENREPORT
                        Full pathname to the Birt runtime 'genReport.sh'
                        utility. Default to /opt/trafMon/bin/genReport.sh,
                        which is typically a symbolic link to the BIRT RunTime
                        installation/ReportEngine/genReport.sh
```

This scripts can be quite explosive (but may be tuned).

It generates a PDF file for every kind of details report (table at top of the script) and for each flow that has data of the corresponding kind. Flow selection is based on IPv4 regular expression pattern(s) configured in a file given as –f option. The match is applied to both address of the list of known flows.

Reports are produced in a hierarchy of directories rooted at `/var/trafMon/reports/mydetails/` when the pattern file is `/etc/trafMon/report/mydetails.genAddrs`.

3.8 SAMPLE SCRIPT FOR (RE-)STARTING TMON_PROBE

This script is typically scheduled by crontab to restart the probe within the minute where it crashes: the father and child processes should be running.

```
$ ./trafMon_probeResurrect.sh.sample
Bad number of arguments
Usage: ./trafMon_probeResurrect.sh.sample '<tmon_probe start cmd>'
                                                    <logfile_pathname>
```

The first argument is typically made of multiple words, hence the need to enclose it in quotes.

ADAPT TO YOUR NEED

```
# Copyright (c) 2020 AETHIS s.a./n.v., Belgium. All rights reserved.
# www.trafmon.org
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# @(#) trafMon_probeResurrect.sh.sample $Id:
391f24d525824f505870131e3e713c2806e0dece $
#

# Detects that (one of the two) tmon_probe processes has/have crashed
#  and restarts the trafMon probe in background
#
# Usage: trafMon_probeResurrect.sh "<tmon_probe start cmd>" <logfile_pathname>
#
# !!! UP TO YOU TO ADAPT !!!

if [ "$#" -eq 2 -a "a$1" != "a" -a "b$2" != "b" ]
then
```

```
  log=$1
  probe_cmd=$2
  # need to match father and child processes with same probe name
  probe_pattern=\
           `echo $probe_cmd |sed -e 's/.*tmon_probe .* /\.\*tmon_probe\.\* /'`

  # Assumption: the log file should belong to the unprivileged user trafmon
  [ -f $log ] || touch $log
  isRoot=false
  [ `whoami` = root ]  && isRoot=true && chown trafmon $log

  var=$(pgrep -f "$probe_pattern" | wc -l)
  if [ "$var" != "2" ]
  then
      pkill -9 -f "$probe_pattern"
      $probe_cmd &>> $log &
      echo "$(date +%Y%m%dT%H%M) trafMon probe RESTARTED" >> $log
  fi
else
  echo "Bad number of arguments" >> /dev/stderr
  echo "Usage: $0 '<tmon_probe start cmd>' <logfile_pathname>" >> /dev/stderr
fi
```

3.9 SAMPLE SCRIPT FOR (RE-)STARTING CENTRAL SERVER DEAMONS

This sample scripts attempts to cover the presence monitoring of any combination of three service daemons of the central server, and (re-)start the missing one within the minute:

- tmon_collector
- mysqld service
- tomcat java service

The arguments decide which of the three are actually looked at.

```
$ ./trafMon_serverResurrect.sh.sample
Bad number of arguments
Usage: ./trafMon_serverResurrect.sh.sample <log_pathname>
             [('' | '<tmon_coll start cmd>') [('' | '<msqld start cmd>')
                                                 ['<tomcat start cmd>']]]
```

The arguments are typically made of multiple words, hence the need to enclose them in quotes.

ADAPT TO YOUR NEEDS

```
# Optionally detects that the tmon_collector process has crashed
#  and restarts the trafMon collector in background
#
# Optionally detects that the mysqld database server has crashed
#  and restarts it
#
# Optionally detects that the Tomcat server Java process has crashed
#  and restarts it
#
# USAGE: trafMon_serverResurrect.sh <logfile_pathname>
#                                   ["" | "<tmon_collector start cmd>"]
#                                   ["" | "<msqld start cmd>"]
#                                   ["<tomcat start cmd>"]
#
# E.g. trafMon_collectorResurrect.sh /var/log/trafMon/resurect.log
#  "/opt/trafMon/bin/tmon_collector -c /opt/trafMon/xml/production.xml
tMon_svr"
#                                                    "" "service tomcat
restart"

if [ "$#" -lt 2 -o "$#" -gt 4 ]
then
  echo "Bad number of arguments" >> /dev/stderr
```

```
  echo "Usage: $0 <log_pathname> [('' | '<tmon_coll start cmd>')">>
/dev/stderr
  echo "   [('' | '<msqld start cmd>') ['<tomcat start cmd>']]]" >>
/dev/stderr
  exit 1
fi

log=$1
coll_cmd=$2
mysqld_cmd=$3
tomcat_cmd=$4

# Assumption: the log file should belong to the unprivileged user trafmon
[ -f $log ] || touch $log
isRoot=false
[ `whoami` = root ]  && isRoot=true && chown trafmon $log

# 1 - tmon_collector
# ##################
#
# Assumption: the tmon_collector is run as user trafmon
#
var=$(pgrep -f "$coll_cmd" | wc -l)
if [ "a$coll_cmd" != "a" -a "$var" = "0" ]
then
  if [ $isRoot = false ]
  then
    $coll_cmd  &>> $log &
    echo "$(date +%Y%m%dT%H%M) tmon_collector process RESTARTED" >> $log
    exit 0 # no permission for mysqld nor tomcat
  else
    su - trafmon -c "$coll_cmd &>> $log &"
    echo "$(date +%Y%m%dT%H%M) tmon_collector process RESTARTED" >> $log
  fi
fi
if [ $isRoot = true ]
  then
  # 2 - mysqld
  # ##########
  # /usr/sbin/mysqld should be running as user mysql
  var=$(ps aux | grep '^mysql .*/usr/sbin/mysqld' |grep -v grep| wc -l)
  if [ "a$mysqld_cmd" != "a" -a "$var" = "0" ]
  then
    $mysqld_cmd &>> $log
    echo "$(date +%Y%m%dT%H%M) mysqld process RESTARTED" >> $log
  fi

  # 3 - tomcat
  # ##########
```

```
  # Tomcat java server should be running as user tomcat
  var=$(ps aux | grep '^tomcat .*java ' |grep -v grep| wc -l)
  if [ "a$tomcat_cmd" != "a" -a "$var" = "0" ]
  then
    $tomcat_cmd &>> $log
    echo "$(date +%Y%m%dT%H%M) tomcat process RESTARTED" >> $log
  fi
fi
```

4. RELEVANT MYSQL STORED PROCEDURES

Only those procedures subject to direct invocation by the tool administrator (maybe via crontab) are presented below.

The best way to run them (especially via crontab) is to invoke the read-only/execute-only database user `tmon_birt`, but avoiding the need to explicitly provide his password (especially on command line).

Hence, for the dedicated tool account '`trafmon`', create a protected private file with following content:

```
trafmon% cat ~/.my.cnf
[mysql]
 user=tmon_birt
 password=xxxx
% chown trafmon:trafmon ~/.my.cnf
% chmod 0400 ~/.my.cnf
```

This way, it suffices to execute the stored procedure via a command like below. Note: procedure are resident in the database `trafMon_template` and take the **database name as first argument**.

```
mysql -e 'CALL trafMon_template.Partition_drop("trafMon",
                                               "ipsz_table_aggr_1m", 90)'
```

4.1 MERGE PASSIVE FTP DATA TRANSFER VOLUME WITH THE ITS FTP SESSION FLOW INSTANCE

Passive FTP data connections have their pair port numbers being random. However, as for active data transfers, they belong logically to the same flow instance as their FTP session control connection.

Hence it is important to merge these volumes before aggregating volumes according the Activity/Location/Host and peer Country:

```
trafMon_template.Update_ftp_data_in_ipcttable("trafMon",
                                              “YYYY-MM-DD”, “YYYY-MM-DD”)
```

with *start day* and *end day* as arguments. NULL data means yesterday.

It can be used to re-compute the aggregation for volume reports explicitly for a given time period in the past; then invoke the following one for the same interval.

4.2 AGGREGATION OF VOLUME DATA FOR THE SYNTHESIS REPORTS

For probe data:

For the probe data, it is important to invoke the previous procedure just before this one.

```
trafMon_template. Aggr_activityvolumetable_first_level ("trafMon",
                                                “YYYY-MM-DD”, “YYYY-MM-DD”)
```

with *start day* and *end day* as arguments. NULL data means yesterday.

For NetFlow data

```
trafMon_template. Aggr_activityvolumetable_netflow_first_level ("trafMon",
                                                 “YYYY-MM-DD”, “YYYY-MM-DD”)
```

with *start day* and *end day* as arguments. NULL data means yesterday.

4.3 REMOVING THE WORKING TABLES FROM ALL TRAFMON DATABASES

The generation of some reports imposes to create working table in a persistent state (multi-session) without being able to automatically remove them.

Those table start with un underscore (_) and can be blindly dropped by a stored procedure:

```
trafMon_template.Drop_working_tables()
```

4.4 SELECTIVELY DROPPING ANCIENT FINE-GRAIN DATA

The per-minute data are only relevant for the latest days or months. The same applies to individual TCP connection records or, even, to individual file transfers.

Care has been taken to partitions the data tables in separate physical partitions covering data about a certain time period. So it suffice to quickly DROP obsolete partitions to perform database housekeeping without heavy DELETE processing.

Following stored procedure could be invoked blindly from crontab, but should be used with care:

```
trafMon_template.Partition_drop("trafMon", "table_name", KEEP)
```

The argument *table_name* is the full name, including the time granularity. The argument KEEP is the last number of days to preserve up to *yesterday*. So it is NOT based on the last data actually present in the database!

You can see approximate sizes of tables via phpMyAdmin. You can also, as root, look at the files (partitions) and sizes, through directory listing the *MySQL_DATADIR*/*DB_NAME*/.

4.5 SUMMARY HEADER OF PROTOCOL DETAILS REPORTS

The report templates of the per-protocol counters views start with a sum-up of the various counters over the requested time span. This is obtained by a call to:

```
trafMon_template.Report_sum("trafMon", "what", _workingtable)
```

Where *what* is one of

- udpct_sum_row
- tcpct_sum_row
- onewayct_sum_row
- ipct_sum_row
- icmpct_sum_row
- ftpct_other_row
- ftpct_session_row
- tcpcon_row

And *workingtable* is a unique name. The summary results are saved in a persistent working table whose name is *workingtable* prepended with an underscore (_). Such kind of table are to be removed by the `trafMon_template.Drop_working_tables()` procedure.

5. CONFIGURATION FILES

5.1 TRAFMON COMMON XML CONFIGURATION

All trafMon probes and the collector9s) share a same XML configuration file, where the specification of every probe, every probe interface and every collector is defined, as well as the Flow Classes, with matching rules and what observations to be made, and the definition of granular flows (what fields are part of the per flow instance identifier).

The syntax, the possible parameters (XML attributes) and their meaning is explained in details by the fully commented DTD file given in appendix (section 6.1.1 below).

An example XML configuration file is presented in appendix (section 6.1.2 below).

Note that, for permitting to produce the volume based synthesis reports, the following class must be present:

```
  <GranularFlow name="uniDirAtProbeIf" >
    <DistinctIf /> <!-- mandatory when Counters, to avoid double records -->
    <DistinctAddr field="srcdst" />
    <DistinctPort field="portpair" portspec="privileged" />
    <GroupBy      field="ipproto"/>
  </GranularFlow>

  <!-- ALL Unidirectional packets (for volumes counting)
       ================================================
  -->
  <FlowClass id="200" name="ALL_packets"
                                    descr="ALL Unidirectional IP Fragments">
    <Measure interval="1min" >
      <Stats verifChksum="bestEffort">
        <PacketCounters for="allFragments"/>
      </Stats>
    </Measure>
    <FlowGrain ref="uniDirAtProbeIf" />
    <Filter>
      <On probe="trafMon-probe" if="p1p1" />
      <On probe="trafMon-probe" if="p1p2" />
      <PacketExpr>
        <AND>
            <Predicate field="src" op="betw"
                                    value="0.0.0.1" value2="255.255.255.254" />
            <Predicate field="dst" op="betw"
                                    value="0.0.0.1" value2="255.255.255.254" />
        </AND>
      </PacketExpr>
    </Filter>
  </FlowClass>
```

Measurements aggregation can be produced at fine time granularity, but not longer than one minute, which is the basis for the finest data granularity in the database and reporting.

5.2 KNOWN IP ADDRESSES CLASSIFICATION

The default pathname of the file is **`/etc/trafMon/ipInfo.ini`**

It must contain all known IP address ranges (or individual values) that are to be assigned a known Activity and known Location (partitioning the own network of the concerned Organisation (Company, Instititution, Administration …).

The format is a series of section whose header is an IP address with or without subnet range, followed by activity=act_name and location=loc_name, optionally country=ctry_name and city=city_name. These last two are not currently not used for known IP.

Note that is also possible to define IP (ranges) without activity nor location, to complement the GeoIP database.

Sample is provided with the distribution package. Example:

```
[10.10.10.0/24]
activity = EXPORT
location = BERLIN
country = Germany
city = Berlin

[10.10.11.0/24]
activity = MSS
location = MADRID
country = Spain
city = Madrid
```

5.3 DATABASE, USERS AND PASSWORDS AND BIRT URL[i]

5.3.1 db.cred

The file **`/etc/trafMon/cred/db.cred` is read-only by the user trafmon.** It contains the MySQL URL, the operational database name and the definition of two users:

- `trafMon_loader_user` is the user with full privileged on all databases whose name start with `trafMon`; it is used for loading the data;
- `trafMon_report_user` is the user with SELECT,EXECUTE only privileges on all databases whose name start with `trafMon`; it is used for producing reports and executing stored procedures (which, by side effect, can create tables and modify the database content)

By convention, the first is named `tmon_db` and the second `tmon_birt`. Their password field contains **rot13** encoded value of the plain text version.

Example:

```
[DatabaseURL]
db_name = trafMon
db_url = unix:///var/lib/mysql/mysql.sock
#db_name = trafMon_2
#db_url = tcp://db_server:3306

[TMonloaderCredentials]
trafMon_loader_user = tmon_db
trafMon_loader_pwd = GuvfVfZlErnq/JevgrgensZbaQOHfreCnffjbeq

[TMonReportCredentials]
trafMon_report_user = tmon_birt
trafMon_report_pwd = GuvfVfZlErnq-BaylgensZbaQOHfreCnffjbeq
```

Because this file contains MySQL username/password pairs, care must be taken that the file is read-only and only accessible by its owner `tomcat` (*this is the typical username and groupname under which the trafMon tool programs and utilities are executed*):

```
root# chown –R apache:apache /etc/trafMon/
root# chmod 0500 /etc/trafMon/cred/
root# chmod 0400 /etc/trafMon/cred/db.cred
```

[i] You can encode a plain password in rot13 or base64 via, for instance, https://cryptii.com/pipes/rot13 and https://cryptii.com/pipes/text-to-base64.

But there are also other places where such information is stored: for the web application menu bar, and for the BIRT report generation.

5.3.2 Web Application Menu Bar

You must adapt the **`/var/www/html/trafMon/php/include.php`**:

Specify the **trafMon_report_user** and the rot13 encoded **trafMon_report_pwd** values taken from your `db.cred` for, respectively, **$username =** and **$username =** fields.

Also adapt **$hostname =** if MySQL is not running on the host as the Apache HTTP server.

Because this file contains a MySQL username/password, care must be taken that the file is read-only and only accessible by its owner `apache` (*the entire `/var/www/html/trafMon` must be owned by `apache:apache`*):

```
root# chown –R apache:apache /var/www/html/trafMon/
root# chmod 0400 /var/www/html/trafMon/php/include.php
```

The Menu Bar is referring to Tomcat/BIRT via the URL prefix saved in **`/var/www/html/trafMon/scripts/services/birturl.js`**:

Following sample pre-supposes HTTPS port 8443 and BIRT accessed via tunnelling to the server using SSH port replication (hence accessed by localhost 120.0.0.1). You should probably at least adapt the IP address with the publicly accessible server hostname

Example:

```
angular.module('trafMonWebApp')
  //.constant('birtUrl', 'http://myserver:8080/birt/');
  .constant('birtUrl', 'https://127.0.0.1:8443/birt/');
```

5.3.3 BIRT Report Template Libraries

All trafMon report templates import their data source definition from a common library:

`/opt/trafMon/trafMon_reports/Library/trafMonDb.rptlibrary` (from the distribution, and used for batch report generation) also copied to **`/var/lib/tomcat/webapps/birt/trafMon_reports/Library/trafMonDb.rptlibrary`** (the web-based report generation and viewing).

Following line must be adapted for mentioning possible hostname where MySQL is running (when not on same host as Tomcat/BIRT):

```
…
            <property name="name">odaURL</property>
            <property name="id">5</property>
            <expression name="value"
type="javascript">'jdbc:mysql://127.0.0.1/'+params["DBname"].value</expression>
…
```

Following line must be adapted for mentioning possible hostname where MySQL is running (when not on same host as Tomcat/BIRT), maybe also the **trafMon_report_user** username (when not `tmon_birt`). And you must replace the password string by the base64 encoding of the database user plain text password.

```
…
            <property name="odaDriverClass">com.mysql.jdbc.Driver</property>
            <property name="odaURL">jdbc:mysql://127.0.0.1/trafMon</property>
            <property name="odaUser">tmon_birt</property>
            <encrypted-property name="odaPassword"
          encryptionID="base64">QUtIcVlmQk1SektWcWpkWg==</encrypted-property>
…
```

Because this file contains a MySQL username/password, care must be taken that the file is read-only and only accessible by its owner `trafmon` or `tomcat` (*the entire `/var/lib/tomcat/webapps/trafMon/` must be owned by `tomcat:tomcat`*):

```
root# chown –R tomcat:tomcat /var/lib/tomcat/
root# chmod 0400
        /var/lib/tomcat/webapps/birt/trafMon_reports/Library/trafMonDb.rptlibrary

root# chown –R trafmon:trafmon /opt/trafMon/
root# chmod 0400 /opt/trafMon/trafMon_reports/Library/trafMonDb.rptlibrary
```

The following file contains the URL necessary to permit trafMon reports to embed a hyperlink to another trafMon report (e.g. Operator to Conversation and the reverse):

`/opt/trafMon/trafMon_reports/Library/url.rptlibrary` also copied to **`/var/lib/tomcat/webapps/birt/trafMon_reports/Library/url.rptlibrary`**

The hostname, maybe also the HTTP/HTTPS selection and the URL port number would need change in:

```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns="http://www.eclipse.org/birt/2005/design" version="3.2.23" id="1">
    <property name="createdBy">Eclipse BIRT Designer Version 4.5.0.v201506092134
Build &lt;@BUILD@></property>
    <property name="units">in</property>
    <property name="theme">defaultTheme</property>
    <parameters>
```

```
        <scalar-parameter name="url" id="7">
            <property name="hidden">true</property>
            <property name="valueType">static</property>
            <property name="isRequired">false</property>
            <property name="dataType">string</property>
            <property name="distinct">true</property>
            <simple-property-list name="defaultValue">
                <value
type="constant">https://127.0.0.1/trafmon/#!/volume</value>
            </simple-property-list>
            <list-property name="selectionList"/>
            <property name="paramType">simple</property>
            <property name="concealValue">true</property>
            <property name="controlType">text-box</property>
            <structure name="format">
                <property name="category">Unformatted</property>
            </structure>
        </scalar-parameter>
        <scalar-parameter name="urlBirt" id="8">
            <property name="hidden">true</property>
            <property name="valueType">static</property>
            <property name="isRequired">false</property>
            <property name="dataType">string</property>
            <property name="distinct">true</property>
            <simple-property-list name="defaultValue">
                <value type="constant">https://127.0.0.1:8443</value>
            </simple-property-list>
            <list-property name="selectionList"/>
            <property name="paramType">simple</property>
            <property name="concealValue">true</property>
            <property name="controlType">text-box</property>
            <structure name="format">
                <property name="category">Unformatted</property>
            </structure>
        </scalar-parameter>
    </parameters>
    <themes>
        <theme name="defaultTheme" id="4"/>
    </themes>
    <page-setup>
        <simple-master-page name="NewSimpleMasterPage" id="3"/>
    </page-setup>
</library>
```

The first URL refers to the Synthesis Reports Menu Bar, the second one to the Tomcat/BIRT.

5.3.4 Avoiding Command Line Explicit Password

When invoking the `mysql` command line client or `mysqladmin` command line tool in an automated way (via `crontab` task or within `logrotate` job), it is not possible to enter the database user password in an interactive way.

It could be passed as command line option `--password=`***`password`*** or `-p`***`password`***. But this plain text code can easily be seen from a process listing.

Hence a more secure way is to write it in a specifically protected file. Unfortunately, only the plain text value is supported.

1. For allowing scheduled `mysql` client execution, via `cron`, of trafMon specific stored procedures by the dedicated Linux account `trafmon`, the following protected file must be created:

```
trafmon$ cat > ~/.my.cnf << EOF
[client]
user=tmon_birt
password=password
EOF
trafmon$ chmod 0400 ~/.my.cnf
```

2. For allowing the root execution of `logrotate` to let the `mysqld` server restart to produce its logging messages to a new file (after rotation), the mysqladmin flush-logs command must be executed. hence the MySQL database administrator username and password must be encoded in a protected file accessible only by root superuser:

```
root# cat > ~/.my.cnf << EOF
[mysqladmin]
user=root
password=password
EOF
root# chmod 0400 ~/.my.cnf
```

5.3.5 Typical Logrotate configuration for MySQL Server

This file must be tuned to your own need. Probably the installation of your MySQL server has already created a file `/etc/logrotate.d/mysql`. Here is a sample with explanations:

```
#
# @(#) mysql.logrotate.sample $Id:$
#
# The log file names and locations can be set in /etc/my.cnf by setting
#     the "log-error" option
#     the "general_log" and "general_log_file" options
# in either [mysqld] or [mysqld_safe] section as
# follows:
```

```
#
# [mysqld]
# log_error=/var/log/mysql/mysqld_err.log
# general_log_file=/var/log/mysql/mysqld_general.log
# general_log=ON
#
# In case the root user has a database password, then you
# have to create a /root/.my.cnf configuration file
#     chown root:root /root/.my.cnf
#     chmod 0400 /root/.my.cnf
# with the following content:
#
# [mysqladmin]
# password = <mysql password for root user>
# user= root
#
# !!! UP TO YOU TO ADAPT !!!
#


/var/log/mysqld/*.log {
        # create 600 mysql mysql
        notifempty
        daily
        rotate 7
        missingok
        compress
        lastaction
          chown -R mysql:mysql /var/log/mysqld/
          if test -x /usr/bin/mysqladmin && /usr/bin/mysqladmin ping &>/dev/null
          then
             /usr/bin/mysqladmin flush-logs
          fi
        endscript
}
```

5.4 DIAGNOSTIC TRACE FILES AND LOGGING

The default diagnostic trace verbosity tuning of the trafMon probes and collector(s) are normally stored under **`/etc/trafMon/diag/`**. Each file name is made by the name of probe or collector instance, as specified in the XML configuration file, followed by **`.diag`**.

Possible levels are: `fatal, error, warning, trace0, trace1, trace2, trace3`

Of course one level means also all the other levels at its left.

For efficiency during production mode, the `Highest_level` disables the message formatting processing for all messages that are above that level, even though they would have been printed out due to per-module specification.

Each module name is the name of a C source code file. It is followed by the maximum level to be printed out and the list of log file pathnames where messages should be written to. Specific output filename are ‘`stdout`’ and (preferably) ‘`stderr`’

The hash mark (#) comments-out the line, so it allows to prepare alternative verbosity tuning, for later switching to.

Example:

```
# Sample of nominal trafMon Probe diagnostic file for operations
#
# Possible levels: fatal, error, warning, trace0, trace1, trace2, trace3
#
Highest_level  trace2

#
# FORMAT
# ======
#   program   module            level     log log ...
#   WHERE log is a full pathname or stdout or stderr
#
tmon_probe tmon_probe         trace0   /var/log/trafMon/myProbe.log stderr

tmon_probe tmon_btree         trace0   /var/log/trafMon/myProbe.log
tmon_probe tmon_circ_buf      trace2   /var/log/trafMon/myProbe.log
tmon_probe tmon_config        trace1   /var/log/trafMon/myProbe.log
tmon_probe tmon_diag          warning  /var/log/trafMon/myProbe.log stderr
tmon_probe tmon_dict          trace1   /var/log/trafMon/myProbe.log
…

#tmon_probe tmp_aggregate      warning  /var/log/trafMon/myProbe.log
tmon_probe tmp_analyse        trace1   /var/log/trafMon/myProbe.log
#tmon_probe tmp_analyse        trace2   /var/log/trafMon/myProbe.log
tmon_probe tmp_child          warning  /var/log/trafMon/myProbe.log
#tmon_probe tmp_child          trace1   /var/log/trafMon/myProbe.log
…
```

Normally, all trafMon log files are produced under **`/var/log/trafMon/`**.

A sample logrotate is provided with the distribution package. It must be copied to **`/etc/logrotate.d/trafMon`** and adapted.

```
# Sample of logrotate specification for the trafMon log files
# Assumptions: all files are produced under /var/log/trafMon/
#              this dectory and all log files are writeable by an
#                           unprivileged user trafmon of group trafmon
#              all these log files end with '.log'
#
# !!! UP TO YOU TO ADAPT !!!
#
```

```
/var/log/trafMon/*.log {
  # When some files have been mistakenly created as root,
  # this can perturbate the automated operations, so restore ownership
  firstaction
  chown -R trafmon:trafmon /var/log/trafMon
  endscript

  lastaction
  chown -R trafmon:trafmon /var/log/trafMon
  endscript

  rotate 300000
  # on CentOS 6.x, use daily insead of not yet suppoprted hourly
  hourly
  size 200M
  compress
  delaycompress
  missingok
  notifempty
  create 0644 trafmon trafmon
}
```

The `root` crontab must also contain a line like:

```
# check every hour to rotate the various logs
0 * * * * /usr/sbin/logrotate /etc/logrotate.d/trafMon
```

5.5 REGULAR TRAFMON JOBS

5.5.1 Typical Tasks

Several tasks have to be regularly executed to maintain the trafMon observations up-to-date and, optionally, to produce batch PDF reports and/or perform data housekeeping and clean-up.

- Load and update aggregates of raw trafMon observations and, optionally, of NetFlow log records; then update the information about not yet resolved IP addresses,

  Every 10 minutes or so;

- Optionally extract newly produced SiLK records with NetFlow/sFlow/IPFIK raw observations (these will then be handled by the above mentioned loader);

  Every hour typically (not possible to have higher frequency with SiLK `rwfilter` command);

- Perform mapping of passive FTP data transfers with their corresponding FTP session flows, then pre-aggregate trafMon flow volumes related information, based on Activity/Location and remote Country partitioning of host addresses,

  Once a day (at night, when database isn't loaded by report generation);

- Optionally, pre-aggregate NetFlow related information, based on Activity/Location and remote Country partitioning of host addresses,

  Once a day (at night, when database isn't loaded by report generation or other task);

- If necessary, and according to user-specified criteria, generate PDF report in batch:
  - Synthesis reports (manager and/or operator and/or conversation templates): general, or for a given Activity and/or Location, or for a given Host,

    Every day or week or month, depending on the requested time span of the report – at a quiet period in the night;

  - Protocol details reports (list is inside the Python script) for every flow with a given set of hosts,

    Every day or week or month, depending on the requested time span of the report – at a quiet period in the night;

- Perform a systematic removal of all working database tables left as side effect of the (on-demand or batch) generation of details reports with counters,

  Once a day (lightweight);

- Carefully perform data housekeeping and clean-up of ancient raw of fine-grain observations (note that log files are handled by their *logrotate* configuration files):
  - Find and remove ancient compressed tarfiles with raw observation from trafMon collector and, optionally, NetFlow extraction;
  - Find and remove ancient data logs from the SiLK tree of log files;
  - Find and remove ancient PDF reports optionally generated in batch;
  - Drop partitions of specified granular database tables with data oolder than a given number of days

- Detect the absence/crash of the `tmon_probe` or `tmon_collector` process and restart it asap,

  Check every minute, as trafmon user;

- Detect the absence/crash of the MySQL database server and/or of the Tomcat service and restart it asap

  Check every minute, as root super user;

- Systematically restart the Tomcat service to recuperate its reserved memory,

Every day – at a quiet period in the night – as root super user;

5.5.2 Probe Systems Crontab Entries

The `crontab` file of the account that executes the `tmon_probe` (either `root` or, thanks to special `setcap` setting, the `trafmon` account) must contain the following task which detects that the probe is not running (with its two nominal processes) and starts it within the minute:

```
## IN THE CRONTAB OF ROOT USER, when the tmon_probe is run as superuser
## OR IN THE CRONTAB OF AN UNPRIVILEGED USER, member of the group trafmon,
##      When following has been applied:
##          # chgrp trafmon /opt/trafMon/bin/tmon_probe
##          # setcap cap_net_raw,cap_net_admin=eip /opt/trafMon/bin/tmon_probe
## DON'T FORGET TO ADAPT YOUR ARGUMENTS TO tmon_probe BELOW
##
## Detects, every minute, whether tmon_probe has crashed and restarts it
* * * * * /bin/bash /opt/trafMon/bin/trafMon_probeResurrect.sh
"/opt/trafMon/bin/tmon_probe TMonProbe" /var/log/trafMon/cron.log
```

Don't forget to adapt the `tmon_probe` execution pattern (first argument between quotes), especially the name of the probe instance.

The `crontab` file of the `root` account must contain the hourly invocation of the rotation of the trafMon log files:

```
## IN THE CRONTAB OF ROOT USER: rotate all trafMon log files
0 * * * * /usr/sbin/logrotate /etc/logrotate.d/trafMon
```

5.5.3 Central Server (trafMon + MySQL + Tomcat) `root` Crontab

Several tasks must be scheduled for the `root` account relative to central services. The assumption below is that all three services run on the same server. But if they are split over two or three different machines, the respective tasks must be defined on the appropriate server.

- On the server running the `tmon_collector`, the log files rotation must be executed every hour in the `crontab` file of the `root` account:

```
## IN THE CRONTAB OF ROOT USER: rotate all trafMon log files
0 * * * * /usr/sbin/logrotate /etc/logrotate.d/trafMon
```

- On the server running the Tomcat service, the `root` account `crontab` file should specify a systematic restart of the service at a quiet period in the night, in order to recuperate the consumed memory:

```
# restarts TOMCAT every night at 1:01:00 to recuperate its memory
1 1 * * * /sbin/service tomcat restart >> /var/log/trafMon/cron.log 2>&1
```

- In the below task, the *second argument* is left empty (it concerns the `tmon_collector` daemon, which runs under an unprivileged `trafmon` account), the *third argument* is the command to (re-)start `mysqld` service (must be left empty, but present with two successive quotes, if MySQL server runs on another system), the *fourth argument* is the command to (re-)start the Tomcat service (not present if Tomcat server is running on another system):

```
# detects, every minute, whether MySQL of TOMCAT has crashed and restarts
* * * * * /bin/bash /opt/trafMon/bin/trafMon_serverResurrect.sh
                                          /var/log/trafMon/resurrect.log ""
                                             "/sbin/service mysqld restart"
                                             "/sbin/service tomcat restart"
```

5.5.4 Crontab for the Central trafMon Collector Account

```
#
# @(#) crontab.trafmon.sample $Id:$
#
# Sample of crontab for the unprivileged trafmon user on the TrafMon server
# which runs the tmon_collector
#
##########################
# environment variables #
##########################
MAILTO=""
PYTHONPATH=$PYTHONPATH:/opt/trafMon/bin/
PATH=$PATH:/opt/trafMon/bin/

# remarks:
#   * python scripts MUST be started in this way
#                                        'python /path/../myScript.py [args]'
#     and not directly '/path/../myScript.py [args]'
#     It is used to determine whether the script is already running or not
#   * grep -v '/bin/sh' is used to detect lines relative to cron.
#     If the shell needs to be changed, change the pattern in the python
#     scripts too.

# m h dom mon dow command

## Stamp new day in cron.log
0 1 * * * date >>/var/log/trafMon/cron.log 2>&1

## Load collector data files into the DB
##    only when trafMon_loader.py returns with status 0,
```

```
##       then trafMon_updateIpInfo.py - partial - is executed to resolve
##                                          IP addresses with no DNS name
*/10 * * * * python /opt/trafMon/bin/trafMon_loader.py -p /var/trafMon/collector
                                                  >>/var/log/trafMon/cron.log 2>&1
   && python /opt/trafMon/bin/trafMon_updateIpInfo.py -p /etc/trafMon/ipInfo.ini
                                                  >>/var/log/trafMon/cron.log 2>&1
#
## Full DNS update of all the IPs info every saturday -- maybe too frequent
59 6 * * sat python /opt/trafMon/bin/trafMon_updateIpInfo.py
                                    -p /etc/trafMon/ipInfo.ini –all
                                      >>/var/log/trafMon/ipInfoFullUpdate.log 2>&1
```

If necessary, adapt the pathname of the **ipInfo.ini** file above.

```
#
#
## Update of ipcttable with re-assigning data bytes of passive FTP transfers to
## the flow with the FTP control connection
##                     (null from/to arguments means do for yesterday)
## THEN
## fill activity/location volume table every day with yesterday's data
##         (must be executed *AFTER* update_ftp_data_in_ipcttable every day)
## the user trafmon must have a $HOME/.my.cnf with permission 0400 for auto
## login as the tmon_birt read-only/call-only user into MySQL:
## % cat ~/.my.cnf
## [mysql]
## user=tmon_birt
## password=xxxx
47 0 * * * mysql -e 'CALL trafMon_template.Update_ftp_data_in_ipcttable(
                     "trafMon", null, null)' >>/var/log/trafMon/cron.log 2>&1 ;
           mysql -e 'CALL trafMon_template.Aggr_activityvolumetable_first_level(
                     "trafMon", null, null)' >>/var/log/trafMon/cron.log 2>&1
```

If necessary, adapt the name of the runtime **trafMon** database above. The double null arguments means that start/end date-time cover “yesterday”.

```
#
#
## Load data from NetFlow SiLK logs every hour
15 * * * * python /opt/trafMon/bin/trafMon_FormatNetFlow.py -H 1
                                                  >> /var/log/trafMon/cron.log 2>&1
```

The above is optional: it collects one hour of NetFlow observations every hour.

```
#
## Fill NetFlow volume table every day
2 3  * * * mysql -e 'CALL
trafMon_template.Aggr_activityvolumetable_netflow_first_level(
                     "trafMon", null, null)' >> /var/log/trafMon/cron.log 2>&1
```

The above is optional: it prepares NetFlow data of “yesterday” for the generation of synthesis reports.

```
#
## Restart a crashed tmon_collector as soon as possible
* * * * * /bin/bash /opt/tmon_bin/trafMon_serverResurrect.sh
                                  /var/log/trafMon/resurrect.log
                                     "/opt/trafMon/bin/tmon_collector TMonServer"
```

Adapt the second (quoted) argument to the exact command line to start the collector, in particular, adapt the collector instance **TMonServer** name to be in line with the XML configuration file (default is */opt/trafMon/xml/tmon.xml*). Compared to section 5.5.3 above, the script is not invoked with third and fourth arguments (about MySQL and Tomcat services).

```
#
## Generate manager report for last month every month (arguments by default)
35 5 1 * * python /opt/trafMon/bin/trafMon_volumeReportGen.py >>
/var/log/trafMon/cron.log 2>&1
## Generate detail protocol reports about myServers for last month every month
#45 5 1 * * python /opt/trafMon/bin/trafMon_detailReportGen.py -f
myServers.genAddrs -s `date -d "yesterday" "+%Y-%m-01"` -t monthly >>
/var/log/trafMon/cron.log 2>&1
```

The above is an example: choose your own quiet schedule at night and the appropriate arguments for the tasks that automatically generate PDF report files with synthesis or details views of specified traffic subset. Note also that the list of details reports is at the start of the `/opt/trafMon/bin/trafMon_detailReportGen.py` and could be edited to suit your needs.

```
#
#
## CLEAN DATABASE of the '_xxx' working tables
34 * * * * mysql -e 'CALL trafMon_template.Drop_working_tables()'
                                                  >> /var/factseo/log/cron.log 2>&1
#
#
## DO THIS WITH CARE: drop the partitions with quickly growing fine grain
##                    data at 1 minute and that are about 90 days ago
##                    e.g. the IP size histograms
## every 3 day at 7:16
#16 7 */3 * * mysql -e 'CALL trafMon_template.Partition_drop(
                                        "trafMon", "ipsz_table_aggr_1m", 90)'
                                                  >> /var/factseo/log/cron.log 2>&1
```

The clean-up of working table is harmless. It is applied to every database with prefix *trafMon.*

But automating the tasks to systematically drop ancient (here older than **90** days) partitions in the given fine-grain details data table (here "**ipsz_table_aggr_1m**") from the run-time database (here named "**trafMon**") requires to beforehand acquire a certain experience with how the database tables are growing and which data tables can afford a systematic blind clean-up. Indeed, re-constituting those dropped data chunks by re-loading the

corresponding raw data files retrieved from the archive can be a tedious job but can also quite (too much) resource hungry ending up to unpredicted crash of the database server due to exhaustion of the available amount of necessary memory! A better way could be to prior keep, offline, a compressed archive of SQL dump of the data subject to clean-up. This way their potential restoration would be lighter (provided you manually recreated to corresponding empty partitions).

5.5.5 Other Disk Growing Data

In fact, the above described cron-based automatic clean-up does not cover all ever growing data on the trafMon central server.

The ASCII logs produced by the `tmon_collector` as well as the SiLK extracted NetFlow records, after pre-processing and merging as early step of the regular loading, are first saved in a compressed tarfile archived under `/var/trafMon/collector/done/` with the name *YYYYMMDD***T***hhmmss***.tbz.** It is up to you to move them to an offline support and to remove them from the disk.

The automatically batch produced PDF reports are saved in a tree hierarchy. Depending on what you systematically generate, this set of files can grow quickly.

The binary logs produced by the optional SiLK `rwflowpack` are accumulating under `/var/silk/`**`data/ext2ext/`***`YYYY/MM/DD`***`/ext2ext-`***`SS_YYYYMMDD.HH`**, where *YYYY* is the year, *MM* is the month, *DD* is the day, *HH* is the hour and *SS* is the SiLK sensor name. It is up to you to remove those ancient NetFlow observations.

As a hint, a good way to remove the tail files (type f), within a hierarchy, that have not been modified since DDD days is given by the command

```
$ find /var/xxx/yyy/zzz –type f –mtime +DDD –delete
```

Recursively removing empty directories, back upward (-depth is implied by –empty), is more complex; anyway these do not occupy space on the disk:

```
$ find /var/xxx/yyy/zzz –type d –empty –exec /usr/bin/rmdir +
```

5.6 SAMPLE CERT® SILK ADD-ON CONFIGURATION

Following is the (old) sample of CERT SiLK `rwflowpack.conf` content provided by the Mark Thomas, lead developer of this open source NetFlow/sFlow/IPFIX toolset, where the values of relevant configuration fields are adapted to our baseline context proposed for trafMon installation.

```
### Packer configuration file  -*- sh -*-
##
## The canonical pathname for this file is
## /usr/local/etc/rwflowpack.conf
##
## RCSIDENT("$SiLK: rwflowpack.conf.in 60e5dccfed7c 2015-09-23 20:32:54Z mthomas
$")
##
## This is a /bin/sh file that gets loaded by the init.d/rwflowpack
## wrapper script, and this file must follow /bin/sh syntax rules.
#
# modified by AETHIS s.a., Belgium as sample for NetFlow/sFlow/IPFIX additional
# source of observations to the open source trafMon utility (www.trafmon.org)
#

# Set to non-empty value to enable rwflowpack
ENABLED=1

# These are convenience variables for setting other values in this
# configuration file; their use is not required.
statedirectory=/var/silk

# If CREATE_DIRECTORIES is set to "yes", the directories named in this
# file will be created automatically if they do not already exist
CREATE_DIRECTORIES=yes

# Full path of the directory containing the "rwflowpack" program
BIN_DIR=/usr/local/sbin

# The full path to the sensor configuration file.  Used by
# --sensor-configuration.  YOU MUST PROVIDE THIS (the value is ignored
# when INPUT_MODE is "respool").
SENSOR_CONFIG=/var/silk/sensor.conf

# The full path to the root of the tree under which the packed SiLK
# Flow files will be written.  Used by --root-directory.
DATA_ROOTDIR=/var/silk/data

# The full path to the site configuration file.  Used by
# --site-config-file.  If not set, defaults to silk.conf in the
# ${DATA_ROOTDIR}.
SITE_CONFIG=/var/silk/twoway-silk.conf

# Specify the path to the packing-logic plug-in that rwflowpack should
# load and use.  The plug-in provides functions that determine into
```

```
# which class and type each flow record will be categorized and the
# format of the files that rwflowpack will write.  When SiLK has been
# configured with hard-coded packing logic (i.e., when
# --enable-packing-logic was specified to the configure script), this
# value should be empty.  A default value for this switch may be
# specified in the ${SITE_CONFIG} site configuration file.  This value
# is ignored when INPUT_MODE is "respool".
PACKING_LOGIC=

# Data input mode.  Valid values are:
#  * "stream" mode to read from the network or from probes that have
#    poll-directories
#  * "fcfiles" to process flowcap files on the local disk
#  * "respool" to process SiLK flow files maintaining the sensor and
#    class/type values that already exist on those records.
INPUT_MODE=stream

# Directory in which to look for incoming flowcap files in "fcfiles"
# mode or for incoming SiLK files in "respool" mode
INCOMING_DIR=${statedirectory}/incoming

# Directory to move input files to after successful processing.  When
# in "stream" mode, these are the files passed to any probe with a
# poll-directory directive.  When in "fcfiles" mode, these are the
# flowcap files.  When in "respool" mode, these are the SiLK Flow
# files.  If not set, the input files are not archived but are deleted
# instead.
##ARCHIVE_DIR=${statedirectory}/archive

# When using the ARCHIVE_DIR, normally files are stored in
# subdirectories of the ARCHIVE_DIR.  If this variable's value is 1,
# files are stored in ARCHIVE_DIR itself, not in subdirectories of it.
FLAT_ARCHIVE=0

# Directory to move an input file into if there is a problem opening
# the file.  If this value is not set, rwflowpack will exit when it
# encounters a problem file.  When in "fcfiles" mode, these are the
# flowcap files.  When in "stream" mode, these are the files passed to
# any probe with a poll-directory directive.
ERROR_DIR=  #${statedirectory}/error

# Data output mode.  As of SiLK-3.6.0, valid values are
# "local-storage", "incremental-files", and "sending".
#
# For compatiblity with previous releases prior to SiLK-3.6.0, "local"
# is an alias for "local-storage" and "remote" and is an alias for
# "sending".
#
# In "local-storage" (aka "local") mode, rwflowpack writes the records
# to hourly files in the repository on the local disk.  The root of
# the repository must be specified by the DATA_ROOTDIR variable.
#
# In "incremental-files" mode, rwflowpack creates small files (called
# incremental files) that must be processed by rwflowappend to create
```

```
# the hourly files.  The incremental-files are created and stored in a
# single directory named by the INCREMENTAL_DIR variable.
#
# In "sending" (aka "remote") mode, rwflowpack also creates
# incremental files.  The files are created in directory specified by
# the INCREMENTAL_DIR variable and then moved to directory specified
# by the SENDER_DIR variable.
OUTPUT_MODE=local-storage

# When the OUTPUT_MODE is "sending", this is the destination directory
# in which the incremental files are finally stored to await
# processing by rwflowappend, rwsender, or another process.
SENDER_DIR=${statedirectory}/sender-incoming

# When OUTPUT_MODE is "incremental-files" or "sending", this is the
# directory where the incremental files are initially built.  In
# "incremental-files" mode, the files remain in this directory.  In
# "sending" mode, the incremental files are moved to the SENDER_DIR
# directory.
INCREMENTAL_DIR=${statedirectory}/sender-incoming

# The type of compression to use for packed files.  Left empty, the
# value chosen at compilation time will be used.  Valid values are
# "best" and "none".  Other values are system-specific (the available
# values are listed in the description of the --compression-method
# switch in the output of rwflowpack --help).
COMPRESSION_TYPE=

# Interval between attempts to check the INCOMING_DIR or
# poll-directory probe entries for new files, in seconds.  This may be
# left blank, and will default to 15.
POLLING_INTERVAL=

# Interval between periodic flushes of open SiLK Flow files to disk,
# in seconds.  This may be left blank, and will default to 120.
FLUSH_TIMEOUT=

# Maximum number of SiLK Flow files to have open for writing
# simultaneously.  This may be left blank, and will default to 64
FILE_CACHE_SIZE=

# Whether rwflowpack should use advisory write locks.  1=yes, 0=no.
# Set to zero if messages like "Cannot get a write lock on file"
# appear in rwflowpack's log file.
FILE_LOCKING=1

# Whether rwflowpack should include the input and output SNMP
# interfaces and the next-hop-ip in the output files.  1=yes, 0=no.
# The default is no, and these values are not stored to save disk
# space.  (The input and output fields contain VLAN tags when the
# sensor.conf file contains the attribute "interface-values vlan".)
PACK_INTERFACES=0

# Setting this environment variable to 1 causes rwflowpack to log the
```

```
# NetFlowV9/IPFIX templates that it receives.
SILK_IPFIX_PRINT_TEMPLATES=

# The type of logging to use.  Valid values are "legacy" and "syslog".
LOG_TYPE=legacy

# The lowest level of logging to actually log.  Valid values are:
# emerg, alert, crit, err, warning, notice, info, debug
LOG_LEVEL=info

# The full path of the directory where the log files will be written
# when LOG_TYPE is "legacy".
LOG_DIR=/var/log/silk/

# The full path of the directory where the PID file will be written
PID_DIR=${LOG_DIR}

# The user this program runs as; root permission is required only when
# rwflowpack listens on a privileged port.
USER=trafmon
#USER=root
#USER=`whoami`  # run as user invoking the script

# Extra options to pass to rwflowpack
EXTRA_OPTIONS=

# Extra environment variables to set when running rwflowpack.  These
# should be specified as VAR=value pairs as shown here:
#EXTRA_ENVVAR='FOO=1 BAR=baz'
EXTRA_ENVVAR=
```

The above implies that you create a directory **`/var/log/silk/`** where the user `trafmon` is allowed to create log file(s) where to write rwflowpack log messages. Note that this could be the common `/var/log/trafMon/`, where all files *xxx*`.log` are under `logrotate` control.

More importantly, you should also create a `trafmon` owned directory `/var/silk/` and a sub-tree rooted at `/var/silk/data/`.

And you must copy the sample `/usr/local/share/silk/twoway-silk.conf` file to `/var/silk/twoway-silk.conf` and adapt this operational with the number and list of sensors (in our case: the different network devices that send us NetFlow or similar data packets) and, maybe, their respective description text.

Finally, create a file **`/var/silk/ sensor.conf`** identifying all the “sensor probes”.

- This file starts with a probes block where each SiLK probe is given a name, a data protocol type (netflow-v5, netflow-v9, sflow, ipfix), a transport protocol (UDP) and destination port number and the source address(es) to accept from:

```
probe S0 netflow-v9
    listen-on-port 9991
```

```
    protocol udp
    accept-from-host 172.25.0.2
end probe

probe S1 netflow-v5
    listen-on-port 7432
    protocol udp
    accept-from-host 10.25.15.6
end probe

probe S2 netflow-v9
    listen-on-port 9991
    protocol udp
    accept-from-host 172.25.1.2
end probe

probe S3 netflow-v9
    listen-on-port 9991
    protocol udp
    accept-from-host 172.25.1.3
end probe
```

- Then follows a sensors block, where each SiLK sensor is given a name, corresponding probe name(s) and type (netflow-v5-probes, netflow-v9-probes, sflow-probes, ipfix-probes), and the respective sides of the source and destination networks.

  In fact, the configuration, for SiLK usage, is more complex and flexible than the one needed for our sole trafMon purpose, where we don't care to whether the observed flows are considered incoming or outgoing: all are simply said "external":

```
sensor S0
    netflow-v9-probes S0
    source-network external
    destination-network external
end sensor

sensor S1
    netflow-v5-probes S1
    source-network external
    destination-network external
end sensor

sensor S2
    netflow-v9-probes S2
    source-network external
    destination-network external
end sensor

sensor S3
    netflow-v9-probes S3
    source-network external
    destination-network external
```

```
end sensor
. . .
```

6. APPENDICES

6.1 TRAFMON PROBE AND COLLECTOR CONFIGURATION

A common XML configuration file is shared between all probes and collector(s) systems. For the production environment, this file is typically called `/etc/tmon/xml/tmon.xml`.

The DTD file `/etc/tmon/xml/tmon.dtd` defines the syntax and is fully commented with the meaning of the parameters.

6.1.1 Configuration Syntax and Explanations: tmon.dtd

```
<!--
      Copyright (c) 2020 AETHIS s.a./n.v., Belgium. All rights reserved.
      www.trafmon.org

      Licensed under the Apache License, Version 2.0 (the "License");
      you may not use this file except in compliance with the License.
      You may obtain a copy of the License at

          http://www.apache.org/licenses/LICENSE-2.0

      Unless required by applicable law or agreed to in writing, software
      distributed under the License is distributed on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
      See the License for the specific language governing permissions and
      limitations under the License.
-->
<!-- trafMon XML DTD tmon.dtd ### Current version $Id: a13d30010506c1a3f0d1e4c04ef9f41149b9c93a $-->
<!ELEMENT TrafMonConfig (Collector+, Probe+,GranularFlow+,
                                                (FlowClass | ReportLink)+) >
<!ATTLIST TrafMonConfig
                    serial          NMTOKEN  #REQUIRED
                    startAt         CDATA    #REQUIRED
                    pktSignBytes    NMTOKEN  "3"
                    maxTravelTime   NMTOKEN  "10000"
                    pduYoungWindow  NMTOKEN  "10"
>
           <!-- serial:      Config. version ID to which every PDU refers  -->
           <!--                 valid values are [0..255], wrapping        -->
           <!-- startAt:     Universal (UTC) Date/Time in sec at which to  -->
           <!--              switch to this TrafMonConfig serial number    -->
           <!--                Format: as per ISO 8601 date representation -->
           <!--                     YYYY-MM-DD hh:mm:ss                    -->
           <!-- pduCRCSize:  how many bytes of PDU content digest [1..3]   -->
           <!-- pduCRCFunc:  hash function for computing PDU content digest-->
           <!--                CURRENTLY ONLY "MD5" is supported           -->
           <!-- pktSignBytes: signature bytes of IP packet content digest  -->
           <!--                 valid values are [2..10]                   -->
           <!-- maxTravelTime: reasonable boundary, in milliseconds, for   -->
           <!--                any packet to travel through the network.   -->
           <!--                Outside this time window, packet of same    -->
           <!--                ReportFlow and same signature are considered-->
           <!--                different.                                  -->
           <!--                Furthermore, after this duration, an        -->
```

```
          <!--                incomplete obs. record, whose missing info  -->
          <!--                are expected from an alive probe, times-out -->
          <!--                in the collector (e.g. packet lost event)   -->
          <!--                 valid values are [100..50000]              -->
          <!-- pduYoungWindow: time window, in seconds in the past,       -->
          <!--                 where the first time in a server received  -->
          <!--                 probe PDU is still considered fresh data   -->
          <!--                 valid values are [3..3600]                 -->
<!ELEMENT Collector       (Addr,Output) >
<!ATTLIST Collector       name      ID                          #REQUIRED
                          ID        NMTOKEN                     #REQUIRED
                          descr     CDATA                       #IMPLIED
                          burstRate NMTOKEN                     "10"
>
          <!-- ID:          [256..]                                       -->
          <!-- burstRate:  how much PDU to expect in a burst              -->
          <!--                      valid values are [10..65535]          -->
<!ELEMENT Addr            EMPTY >
<!ATTLIST Addr            ip        NMTOKEN                     "0.0.0.0"
                          port      NMTOKEN                     #REQUIRED
                          UDPBufferSize
                                    NMTOKEN                     "0"
>
          <!-- ip:               IP on which to bind to receive PDUs     -->
          <!-- port:             port number on which to listen          -->
          <!-- UDPBufferSize:    kernel buffer size, in kilobytes,       -->
          <!--                                       of UDP socket       -->
          <!--                     valid values are [0..65535]           -->
          <!--                      0 means the kernel default max size  -->
<!ELEMENT Output          EMPTY >
<!ATTLIST Output          dataFile     CDATA          "tmondata-%y%m%d%H%M%S"
                          eventFile    CDATA          "tmonevent-%y%m%d%H%M%S"
                          excepFile    CDATA          "tmonexcep-%y%m%d%H%M%S"
                          period       NMTOKEN                     #IMPLIED
>
          <!-- dataFile:         CSV-file radix name for data output     -->
          <!--                      supports strftime strings (%H%M...)  -->
          <!-- eventFile:        CSV-file radix name for event output    -->
          <!--                      supports strftime strings (%H%M...)  -->
          <!-- excepFile:        CSV-file radix name for exceptions output-->
          <!--                      supports strftime strings (%H%M...)  -->
          <!-- period:           number of minutes during which output   -->
          <!--                   accumulate in a same CVS-file           -->
          <!--                      ONLY VALID where strftime %M present -->
          <!--                      Used as the modulo on the minute field-->
          <!--                     valid values are [1..59]              -->

<!ELEMENT Probe           ((CapFile | Interface+), PDUSending*, PDUSaving?) >
<!ATTLIST Probe           name         ID                          #REQUIRED
                          ID           NMTOKEN                     #REQUIRED
                          descr        CDATA                       #IMPLIED
>
          <!-- ID:          [0..255]                                      -->

<!ELEMENT CapFile        EMPTY >
<!ATTLIST CapFile         filename     CDATA                       #REQUIRED
                          ID           NMTOKEN                     #REQUIRED
                          expr         CDATA                       #IMPLIED
                          rate         (withDelay|fullSpeed)       "withDelay"
>
          <!-- filename:   Full pathname of packet capture file to read  -->
          <!-- id:         TrafMon-wide unique numeric ID of the probe   -->
          <!--             interface that can distinguish among granular -->
          <!--             flow instances                                -->
          <!--                    valid values are [1..65535]            -->
          <!-- expr:       tcpdump-like packet capture filter expression -->
          <!--                  WHEN VLAN PARTLY TAGS PRESENT            -->
```

```
            <!--                          INVOLVE vlan at end of expr        -->
            <!--                       MATCH only IP packets                 -->
            <!--                       DON'T use netmask based criteria      -->
            <!-- rate:       fullSpeed: captured pakets are processed       -->
            <!--                        untouched (with their original capt.-->
            <!--                        time) without waiting between each  -->
            <!--                withDelay: every packet has its capture time   -->
            <!--                        artificially translated by a fixed  -->
            <!--                        amount of time so as if the exact   -->
            <!--                        same traffic behaviour would occur  -->
            <!--                        'now'. The necessary variable delay -->
            <!--                        is respected before processing each -->
            <!--                        next packet                         -->

<!ELEMENT Interface        EMPTY >
<!ATTLIST Interface        name          NMTOKEN                    #REQUIRED
                           ID            NMTOKEN                    #REQUIRED
                           descr         CDATA                      #IMPLIED
                           snapLen       NMTOKEN                    "1600"
                           bufPacketCount
                                         NMTOKEN                    "70000"
                           expr          CDATA                      #IMPLIED
>
            <!-- id:          TrafMon-wide unique numeric ID of the probe    -->
            <!--              interface that can distinguish among granular  -->
            <!--              flow instances                                 -->
            <!--                      valid values are [1..65535]            -->
            <!-- snapLen:     maximum Ethernet frame Captured portion        -->
            <!--                      valid values are [125..65535]          -->
            <!--   On Linux:                                                 -->
            <!--       snapLen=125 leads to 122 Eth capture => IP len=108    -->
            <!--                                                             -->
            <!--   NOTE:                                                     -->
            <!--       Even on Ethernet (max MTU = 1500 bytes IP),           -->
            <!--       larger packets can be actually captured due to        -->
            <!--       reassembly being offloaded in the NIC Card            -->
            <!--       Linux Ethtool -k:  LRO - Large Receive Offload or     -->
            <!--                          or GRO - Generic Receive Offload   -->
            <!--       ATTEMPT IS MADE TO DEACTIVATE THIS and the Reception -->
            <!--       Checksum processing offload upon initiatisation of    -->
            <!--       the probe capture interfaces                          -->
            <!--                                                             -->
            <!-- bufPacketCount:                                           -->
            <!--              How many packets (of ~ snapLen) could be       -->
            <!--              buffered upon traffic burst                    -->
            <!--                      valid values are [1000..1000000]       -->
            <!--                                                             -->
            <!-- expr:        tcpdump-like packet capture filter expression -->
            <!--                     WHEN VLAN PARTLY TAGS PRESENT           -->
            <!--                          INVOLVE vlan at end of expr        -->
            <!--                     MATCH only IP packets                   -->
            <!--                     DON'T use netmask based criteria        -->
            <!--     LIMITATION: care must been taken with VLAN packets:     -->
            <!--                 using vlan in expression let the software   -->
            <!--                 use a four bytes offset                     -->
            <!--                 BUT for mixed tagged and untagged traffic   -->
            <!--                 the expression "vlan or udp or tcp" could   -->
            <!--                 fail to work on some systems/NICs           -->

<!ELEMENT PDUSending       (SendTo+) >
<!ATTLIST PDUSending       probeIP       NMTOKEN                    "0.0.0.0"
                           probePort     NMTOKEN                    #REQUIRED
>
            <!-- probeIP:         Source IP for sending PDUs                -->
            <!--                     0.0.0.0 means ANY SOURCE ADDR          -->
            <!-- probePort:       Source port for sending PDUS              -->
<!ELEMENT SendTo           EMPTY >
```

```
<!ATTLIST SendTo            collector     IDREF                     #REQUIRED
                            maxPDUSize    NMTOKEN                   "300"
                            maxPDUBuildTime
                                          NMTOKEN                   "300"
                            minTimeGap    NMTOKEN                   "100"
                            heartBeatDelay
                                          NMTOKEN                   "10"
                            timeout       NMTOKEN                   "10"
                            TOMult        NMTOKEN                   "1"
                            TOIncr        NMTOKEN                   "0"
                            retries       NMTOKEN                   "2"
                            breakBorderTime
                                          NMTOKEN                   "3"
                            dropObsFinalTimeout
                                          NMTOKEN                   "60"
>
            <!-- collector:       Name of target Collector                  -->
            <!-- maxPDUSize:      maximum UDP payload, in bytes, of a PDU   -->
            <!--                     valid values are [200..1460]           -->
            <!-- maxPDUBuildTime: maximum duration, in seconds, that a PDU  -->
            <!--                  under construction waits for new records  -->
            <!--                  before being sent                         -->
            <!--                     0 means to send each record in its own -->
            <!--                       PDU, as soon as published            -->
            <!--                     valid values are [0..65535]            -->
            <!-- minTimeGap:      least gap, in msec, between two           -->
            <!--                                              successive PDU -->
            <!--                     valid values are [0..10000]           -->
            <!-- heartBeatDelay:  max silence delay, in sec, before         -->
            <!--                  publishing a possibly empty PDU to server -->
            <!--                     valid values are [1..600]              -->
            <!-- timeout:         first PDU ack timeout in sec              -->
            <!--                     valid values are [1..120]              -->
            <!-- TOMult:          how many times previous ack timeout at    -->
            <!--                     next retry?                            -->
            <!--                     valid values are [1..10]               -->
            <!-- TOIncr:          secs to add to                            -->
            <!--                           (previous ack timeout * TOMult)  -->
            <!--                     valid values are [0..120]              -->
            <!--                                                            -->
            <!-- FOR TYPES OF OBSERVATIONS SUBJECT TO BE DISCARDED:         -->
            <!-- retries:         (maximum PDU send tries) minus 1          -->
            <!--                    ALSO threshold for detecting loss of    -->
            <!--                                 probe connectivity condition -->
            <!--                     valid values are [0..100]              -->
            <!-- breakBorderTime: time window, in sec, from detection       -->
            <!--                   of probe loss of connectivity, whose     -->
            <!--                   packet observations PDU's are            -->
            <!--                   continuously retried                     -->
            <!--                     valid values are [0..300]              -->
            <!-- dropObsFinalTimeout: probe will anyway discard its         -->
            <!--                      long retried obs after this quite     -->
            <!--                      long delay in MINUTES                 -->
            <!--                     valid values are [1..12000]            -->
            <!--                                                 up to 1 week -->
            <!-- NOTE: retries and breakBorderTime do NOT apply to          -->
            <!--             - Flow Instance Description PDU type           -->
            <!--       which are always retried until PDU contents expire   -->

<!ELEMENT PDUSaving         EMPTY >
<!ATTLIST PDUSaving         filepathname CDATA                      #REQUIRED
                            maxPDUSize   NMTOKEN                    "3000"
>
            <!-- filepathname:    Full pathname of file radix where to      -->
            <!--                  locally save the various types of probe   -->
            <!--                  observation PDU's                         -->
            <!--                     supports strftime strings (%H%M...)    -->
```

```
<!-- FACTS-EO FlowClass covers a new TrafMon behaviour where the traffic   -->
<!-- can be independently measured at the different probe/interfaces, and  -->
<!-- the measurements are selectively aggregated and centralised, according-->
<!-- to  Measure directives associated to different FlowClasses.           -->
<!--                                                                       -->
<!-- A FlowClass is assigned one or more Filter expressions to designate to-->
<!-- which sets of packets and protocol exchanges its directives pertain.  -->
<!-- A given Filter Expression is applied On one or more probe/interfaces  -->
<!--                                                                       -->
<!-- Measurements for a FlowClass are segregated per GranularFlow, before  -->
<!-- being centralised to a Server.                                        -->
<!-- The Criteria to discover and discriminate the Granular Flows can be   -->
<!-- applied via <FlowGrain> at the level of a FlowClass, or specialised   -->
<!-- for specific Filter Expression.                                       -->
<!--                                                                       -->
<!-- A given packet can well match several FlowClasses. Hence different    -->
<!-- sets of measurement directives can apply to a same packet and protocol-->
<!-- exchange.                                                             -->

<!ELEMENT GranularFlow    (DistinctIf?,DistinctAddr?,DistinctPort?,GroupBy*) >
<!ATTLIST GranularFlow    name        ID                          #REQUIRED
>
<!ELEMENT DistinctIf      EMPTY >
            <!-- Packets seen at different Probe Interfaces lead to        -->
            <!-- instances of granular flow, even when they produce same   -->
            <!-- results for all other criteria.                           -->
            <!-- NOTE:                                                     -->
            <!--      This may NOT be used when matching BI-DIRECTIONAL    -->
            <!--      traffic flow on the basis of packets captured by a   -->
            <!--      PASSIVE TAP devices: each direction being seen by a  -->
            <!--      separate capture interface.                          -->
<!ELEMENT DistinctAddr    EMPTY >
<!ATTLIST DistinctAddr    field       (src|srcnet|dst|dstnet|srcdst|srcdstnet
                                       |addr|net|addrpair|netpair) #REQUIRED
                          mask        CDATA                       #IMPLIED
>
            <!-- field:  which fields to preserve in grouping measurements  -->
            <!--  a) UNI-DIRECTIONAL                                        -->
            <!--   src:      keep granularity per source IP address         -->
            <!--   srcnet:   keep granularity per src IP subnet: using mask -->
            <!--   dst:      keep granularity per destination IP address    -->
            <!--   dstnet:   keep granularity per dst IP subnet: using mask -->
            <!--   srcdst:   keep granularity per source/dest. IP addresses -->
            <!--   srcdstnet:keep granularity per src/dst IP subnets: mask  -->
            <!--  b) BI-DIRECTIONAL                                         -->
            <!--   addr:     keep granularity per IP address of 1 peer      -->
            <!--   net:      keep gran. per IP subnet of 1 peer: using mask -->
            <!--   addrpair: keep granularity per pair of IP addresses      -->
            <!--   netpair:  keep granul. per pair of IP subnets: using mask-->
            <!--                                                            -->
            <!-- mask:   subnet mask: "xxx.xxx.xxx.xxx" or "/yy" notation   -->

<!ELEMENT DistinctPort    EMPTY >
<!ATTLIST DistinctPort    field       (sport|dport|sdport
                                       |port|portpair)            #REQUIRED
                          portspec    (alldistinct|privileged)  "alldistinct"
>
            <!-- field:  which fields to preserve in grouping measurements  -->
            <!--  a) UNI-DIRECTIONAL                                        -->
            <!--   sport:    keep granularity of source UDP/TCP port number -->
            <!--                     <=> any:port to any:any                -->
            <!--   dport:    keep granularity of destin. UDP/TCP port number-->
            <!--                     <=> any:any to any:port                -->
            <!--   sdport:   keep granularity of src/dst UDP/TCP prt numbers-->
            <!--                     <=> any:port1 to dst:port2             -->
            <!--    EITHER  without <DistinctAddr>                          -->
            <!--    OR ONLY with    <DistinctAddr field=(src|srcnet         -->
```

```
            <!--                                              and/or dst|dstnet) -->
            <!--   port: EITHER  with <DistinctAddr field=(src|srcnet) >    -->
            <!--               same as sport:  <=> src:sport to any:any     -->
            <!--         OR      with <DistinctAddr field=(dst|dstnet) >    -->
            <!--               same as dport:  <=> any:any to dst:dport     -->
            <!--         OR      with <DistinctAddr field=(srcdst[net]) >   -->
            <!--               preserves smallest port number:              -->
            <!--                 if sport <= dport                          -->
            <!--                    <=> src:sport to dst:any                -->
            <!--                 if sport >  dport                          -->
            <!--                    <=> src:any to dst:dport                -->
            <!--   portpair: ONLY with    <DistinctAddr field=(src|srcnet   -->
            <!--                                              and/or dst|dstnet) -->
            <!--               same as sdport: <=> src:sport to dst:dport   -->
            <!--  OTHER COMBINATIONS of <DistinctAddr>+<DistinctPort>       -->
            <!--                        ARE NOT ALLOWED (and meaningless)   -->
            <!--  b) BI-DIRECTIONAL                                         -->
            <!--   port: EITHER without <DistinctAddr>                      -->
            <!--               keep granul. of UDP/TCP port number of 1 peer  -->
            <!--                 if sport <= dport                          -->
            <!--                    <=> any:sport to any:any                -->
            <!--                 if sport >  dport                          -->
            <!--                    <=> any:dport to any:any                -->
            <!--         OR with <DistinctAddr field=(addr|net) >           -->
            <!--               keep granul.  addr:port  to/from   any:any   -->
            <!--                              net:port  to/from   any:any   -->
            <!--                                 WHERE addr/net <= peer any -->
            <!--         OR with <DistinctAddr field=(addrpair|netpair) >   -->
            <!--               keep granul. of address pair and smallest port:-->
            <!--                 if port1 <= port2                          -->
            <!--                           addr1:port1 to/from addr2:any    -->
            <!--                            net1:port1 to/from  net2:any    -->
            <!--                 if port1  > port2                          -->
            <!--                           addr1:any   to/from addr2:port2  -->
            <!--                            net1:any   to/from  net2:port2  -->
            <!--   portpair:                                                -->
            <!--         EITHER without <DistinctAddr>                      -->
            <!--               keep granul. of both UDP/TCP port numbers    -->
            <!--                     <=> any:port1 to/from any:port2        -->
            <!--                              WHERE port1 <= port2          -->
            <!--         OR with <DistinctAddr field=(addr|net) >           -->
            <!--               keep granul.  addr:port1 to/from   any:port2 -->
            <!--                              net:port1 to/from   any:port2 -->
            <!--                                 WHERE addr/net <= peer any -->
            <!--         OR with <DistinctAddr field=(addrpair|netpair) >   -->
            <!--               keep granul. addr1:port1 to/from addr2:port2 -->
            <!--                              net1:port1 to/from  net2:port2 -->
            <!--                                                            -->
            <!-- portspec:                                                  -->
            <!--  *alldistinct:  keep all values distinct                   -->
            <!--   privileged:  distinguish all service ports<1024        -->
            <!--                  BUT group all ports>=1024 (as 65535)      -->

<!ELEMENT GroupBy          EMPTY >
<!ATTLIST GroupBy          field       (ipsizes
                                       |ipproto|tos|df|mf|frag|ttl
                                       |icmp
                                       |tcptype)                  #REQUIRED
                           sizeclasses (per200|per400)            #IMPLIED
                           tosspec     (precedence|dscp|tosbyte)  #IMPLIED
                           fragspec    (fragnumber|fragoffset)    #IMPLIED
                           icmpspec    (icmpclass|icmptype
                                       |icmptypecode)             #IMPLIED
                           tcptypespec (byflags|byflagsandretran
                                       |S_D_A_E|S_D_A_E_R
                                       |S_F_R_A_E|S_F_R_A_E_R)   "S_D_A_E"
>
```

```
            <!--   ipsizes:  keep granularity per 'sizeclasses' of IP pkt   -->
            <!--   ipproto:  keep granularity per UDP|TCP|Other IP protocol -->
            <!--   tos:      keep granularity as per IP TypeOfSvc 'tosspec' -->
            <!--   df:       keep granularity per IP Don't Fragment flag    -->
            <!--   mf:       keep granularity per IP More Fragment flag     -->
            <!--   frag:     keep granularity as per IP Fragment 'fragspec' -->
            <!--   ttl:      keep granularity per IP Time-to-Live value     -->
            <!--   icmp:     keep granularity of ICMP pkts as per 'icmpspec'-->
            <!--   tcptype:  keep granularity as per 'tcptypespec' grouping -->
            <!--                                                            -->
            <!-- sizeclasses: groups IP packet sizes in buckets             -->
            <!--   per400:   4 buckets: boundaries 400, 800, 1200           -->
            <!--   per200:   8 buckets:     200,400,600,800,1000,1200,1400  -->
            <!-- BUT, for datagram cummulated IP sizes, sizes >= 1600       -->
            <!-- are grouped by thousands: 1600, 2000, 3000 ... 7000, 8000  -->
            <!--                                                            -->
            <!-- tosspec:                                                   -->
            <!--   precedence: per value of the three ToS precedence bits   -->
            <!--   dscp:       per value of the six DSCP bits               -->
            <!--   tosbyte:    per distinct values of the complete ToS byte -->
            <!--                                                            -->
            <!-- fragspec:                                                  -->
            <!--   fragnumber: per ordinal number of the fragment           -->
            <!--   fragoffset: per value of the fragment offset             -->
            <!--                                                            -->
            <!-- icmpspec:                                                  -->
            <!--   icmpclass:    group as per Echo | Error | Info | Other   -->
            <!--   icmptype:     per value of the ICMP Type byte            -->
            <!--   icmptypecode: per value of the ICMP Type and Code bytes  -->
            <!--                                                            -->
            <!-- tcptypespec: distinguish                                   -->
            <!--   byflags:      per distinct values of the TCP flags byte  -->
            <!--   byflagsandretran: idem, but also distinguish between     -->
            <!--                     first and subsequent transmissions     -->
            <!--                     of a not empty data segment            -->
            <!--  *S_D_A_E:      Start (syn/syn-ack),                       -->
            <!--                 Data (not empty payload)                   -->
            <!--                 Ack (ack flag, but no payload)             -->
            <!--                 End (fin/fin-ack/reset)                    -->
            <!--   S_D_A_E_R:    Start (syn/syn-ack),                       -->
            <!--                 Data (not empty payload)                   -->
            <!--                 Ack (ack flag, but no payload)             -->
            <!--                 End (fin/fin-ack)                          -->
            <!--                 RESET (rst flag)                           -->
            <!--   S_F_R_A_E:    Start (syn/syn-ack),                       -->
            <!--                 FIRST transmission of data segment         -->
            <!--                 RETRANSMISSION of data segment             -->
            <!--                 Ack (ack flag, but no payload)             -->
            <!--                 End (fin/fin-ack/reset)                    -->
            <!--   S_F_R_A_E_R:  Start (syn/syn-ack),                       -->
            <!--                 FIRST transmission of data segment         -->
            <!--                 RETRANSMISSION of data segment             -->
            <!--                 Ack (ack flag, but no payload)             -->
            <!--                 End (fin/fin-ack)                          -->
            <!--                 RESET (rst flag)                           -->

<!ELEMENT FlowClass       (Measure, FlowGrain?, Filter+, Condition?)>
<!ATTLIST FlowClass       id           NMTOKEN                  #REQUIRED
                          name         ID                       #REQUIRED
                          descr        CDATA                    #IMPLIED
>

<!ELEMENT Measure         (Delay?, Stats?) >
<!ATTLIST Measure         interval     (each|10s|20s|30s
                                       |1min|10min|20min|30min
                                       |1h)                     #REQUIRED
>
```

```
            <!--    Maximum 1 <Delay>, maximum 1 <Stats>, but not empty      -->

<!ELEMENT Delay          ((OneWayDelay|RoundTripDelay|InterPacket),Histogram?)>
<!ATTLIST Delay           for           (firstFragment|allFragments
                                         |datagram)                #REQUIRED
                          granularity   (individual|collectorAggregated
                                         |probeAggregated)         #REQUIRED
>
<!-- How delay measurements are processed:                                  -->
<!-- ====================================                                  -->
<!--   Either individual measurements are loaded in the database, or these  -->
<!--   are pre-aggregated into histogram slices, either by the collector or -->
<!--   where applicable, by the probe itself, before transmission.          -->
<!--                                                                        -->
<!--              granularity: Where individual measurements are            -->
<!--                           aggregated:                                  -->
<!--                individual    Individual measured values are delivered  -->
<!--                              by the probe to the collector and from the -->
<!--                              collector to the database.                -->
<!--                              So actual aggregation occurs inside the   -->
<!--                              database itself.                          -->
<!--                collectorAggregated                                     -->
<!--                              Individual measured values are delivered  -->
<!--                              by the probe to the collector and these are-->
<!--                              aggregated inside the collector which     -->
<!--                              supplies the database with short duration -->
<!--                              histogram slices                          -->
<!--                probeAggregated                                         -->
<!--                              Individual measured values are aggregated -->
<!--                              inside the probe which transmits resulting -->
<!--                              short duration histogram slices to the    -->
<!--                              collector, in turn supplying them to the  -->
<!--                              database                                  -->
<!--                 ==> probeAggregated NOT VALID for Delay type="oneway"  -->
<!--                 ==> probeAggregated IGNORED for Measure interval="each" -->
<!--                                                                        -->
<!--              interval:    Length of the histogram slice aggregating    -->
<!--                           the measurements                             -->
<!--                each          Used when data are not aggregated         -->
<!--                              (granularity=='individual'), meaning that -->
<!--                              any new value has to be transmitted as soon-->
<!--                              as computed and individually supplied by  -->
<!--                              the collector to the database.            -->
<!--                (duration)    Over which duration measurements have to be-->
<!--                              pre-aggregated (in the probe or collector) -->

<!-- Measuring One-Way Latencies:                                           -->
<!-- ============================                                          -->
<!--   Means that probes sends individual timestamps for data units to the  -->
<!--   central collector(s)                                                 -->
<!--                                                                        -->
<!--     ==> Potentially high volume of observations need to be centralised -->
<!--         in the collector                                               -->
<!--     ==> Implies granularity=individual or collectorAggregated          -->
<!--                                                                        -->
<!--              for:       Which data unit to measure?                    -->
<!--                firstFragment One capture timestamp for single          -->
<!--                              or first IP fragment of a datagram/segment -->
<!--                              ==> IP reassembly is not required for this -->
<!--                                  (second and subsequent fragments are  -->
<!--                                                      ignored for this) -->
<!--                allFragments  One capture timestamp for every fragment  -->
<!--                              ==> only meaningful when NO fragmentation -->
<!--                                        between concerned probing points -->
<!--                              ==> IP reassembly is required for         -->
<!--                                    FlowClass membeship determination   -->
<!--                  + APPLIES ALSO   to subseq. fragments not individually -->
```

```
<!--                                  matching Filter                          -->
<!--                + DOES NOT APPLY to subseq. fragments explicitly           -->
<!--                                  rejected by StatePred Condition          -->
<!--                                                                           -->
<!--              datagram      One capture timestamp for unfragmented pkt     -->
<!--                            But two capture timestamps for chains of       -->
<!--                            datagram fragments: {first seen, last seen}    -->
<!--                            ==> IP reassembly is required for this         -->
<!--                            ==> Latency                                    -->
<!--                                  = lastTS(dst side) -firstTS(src side)    -->
<!--                                                                           -->
<!ELEMENT OneWayDelay     (Hop+, Sign?) >

<!ATTLIST OneWayDelay     from         NMTOKEN                    #IMPLIED
                          to           NMTOKEN                    #IMPLIED
                          lost         (each|count)               "each"
>
            <!-- Producing One-Way Latency:                                    -->
            <!-- ==========================                                   -->
            <!-- When Delay granularity=collectorAggregated                    -->
            <!-- produce the delay between the 'from' Hop and the 'to' Hop     -->
            <!--                                                               -->
            <!-- When Delay granularity=individual                             -->
            <!-- 'from' and 'to' are disregarded                               -->
            <!-- Individual HOP timestamps records are produced                -->
            <!--                                                               -->
            <!-- Note that Delay granularity=probeAggregated IS NOT VALID      -->
            <!--                                                               -->
            <!-- lost "each" : produce individual pkt/datagram lost records    -->
            <!--      "count": produce count of lost data units per given      -->
            <!--               Measure interval unit                           -->
<!ELEMENT Hop EMPTY >
<!ATTLIST Hop             name         NMTOKEN                    #REQUIRED
                          descr        CDATA                      #IMPLIED
>
            <!-- Measuring One-Way Latencies:   hopName per timestamp          -->
            <!-- ============================   ====================          -->
            <!-- NOTE: <Hop> inside a <OneWayDelay> tag are ordered!!          -->
            <!-- The order represents the chronology of each timestamp in      -->
            <!-- the sequence, whatever its type (capture/IP/NTP)              -->
            <!--                                                               -->
            <!-- This tag must be in one or more copies in order to            -->
            <!-- specify the sequence of hop timestamps expected for           -->
            <!-- completing a valid record.                                    -->
            <!-- More than one probe (interface) can provide the same time     -->
            <!-- in the case of redundant transmission via more than one       -->
            <!-- link. But in any way, identical timestamp IDs are             -->
            <!-- expected to identify the same logical hop along the path.     -->
            <!-- If a probing point is labeled 'dmz', two different probes     -->
            <!-- reporting times for 'dmz' are supposed to be located in       -->
            <!-- alternate instances of the corporate DMZ.                     -->
            <!-- Every Hop name is assigned inside one or more <Filter>:       -->
            <!--  + either as the capture timestamp at the interface           -->
            <!--  + or as an IP option timestamp of a given sequence number    -->
            <!--  + or as an NTP timsetamp of a given type from either the     -->
            <!--    NTP Request or NTP Response                                -->

<!ELEMENT Sign     (Mask*, Chunk*) >
<!ELEMENT Mask EMPTY >
<!ATTLIST Mask      field              (srcAddr|dstAddr|srcPort|dstPort
                                        |ipFragData|ipTOS|ipID|ipOpts
                                        |transportCkSum)          #REQUIRED
>
            <!-- There can be multiple fields being masked.                    -->
            <!--                                                               -->
            <!-- field:       which part(s), if any, of the packet to          -->
            <!--              zeroize before signature hash computation.       -->
```

```
            <!--                                                              -->
            <!--              Masking any of srcAddr, dstAddr, srcPort,       -->
            <!--              dstPort also implies masking transportCkSum     -->
            <!--                                                              -->
            <!-- Mask directive can be combined with Chunk directives:        -->
            <!--              Specified fields laying into selected Chunks    -->
            <!--              do not participate to the signature hashing     -->
            <!-- Absence of Mask directive leads to masking only the          -->
            <!-- systematically (TTL, cksum) - or supposedly (ipOpts) -       -->
            <!-- varying fields of the IP header                              -->

<!ELEMENT Chunk EMPTY >
<!ATTLIST Chunk     start             NMTOKEN                    "0"
                    relTo             (ipHeader|ipPayload|tcpHeader|tcpPayload
                                       |udpHeader|udpPayload)    "ipHeader"
                    length            NMTOKEN                    "0"
>
            <!-- There can be one or multiple disjoint chunks of bytes       -->
            <!-- being concatenated then hashed to produce the signature     -->
            <!--                                                              -->
            <!-- By default (absence of Chunk directive), the entire packet  -->
            <!-- is subject to hashing, starting start the IP header         -->
            <!-- (masking at least systematically varying fields of IP hdr)  -->
            <!--                                                              -->
            <!-- When <Delay type="oneway" for="datagram"                     -->
            <!--                       or for="firstFragment"> is specified  -->
            <!--      a fragmented datagram is reassembled, and the IP        -->
            <!--      header, only with common stable fields, being appended-->
            <!--      the reassembled IP payload is subject to signature,     -->
            <!--      according to specified Mask/Chunk directives.           -->
            <!--      ==> length==0 means to the end of reassembled payload   -->
            <!--                                                              -->
            <!-- start: offset form the given relTo base.                     -->
            <!--      ==0 by default                                          -->
            <!--      valid values [0..70000]  (max reass. dgram = 65535)     -->
            <!--                                                              -->
            <!-- relTo: when start==0 (or absent):                            -->
            <!--         ipHeader:   Starts at first byte of IP header        -->
            <!--                                                              -->
            <!--         ipPayload|tcpHeader|udpHeader:                       -->
            <!--                     Starts at first byte after IP header     -->
            <!--                     and IP options                           -->
            <!--                   VALID FOR ANY PROTOCOL ABOVE IP            -->
            <!--                   VALID FOR ANY IP FRAGMENT INDIV. MEASURED-->
            <!--                                                              -->
            <!--         tcpPayload: Starts at first byte after TCP header    -->
            <!--                     and TCP options                          -->
            <!--                   IGNORED FOR NON-TCP PACKETS                -->
            <!--                   IGNORED FOR NON-FIRST IP FRAGMENTS         -->
            <!--                                        INDIVIDUALLY MEASURED -->
            <!--                                                              -->
            <!--         udpPayload: Starts at first byte after TCP header    -->
            <!--                   IGNORED FOR NON-UDP PACKETS                -->
            <!--                   IGNORED FOR NON-FIRST IP FRAGMENTS         -->
            <!--                                        INDIVIDUALLY MEASURED -->
            <!--                                                              -->
            <!-- length:      how many bytes, since the specified start of    -->
            <!--              the chunk, do participate to the pkt signature-->
            <!--              hash computation                                -->
            <!--      valid values [0..70000]  (max reass. dgram = 65535)     -->
            <!--              0 MEANS TO END OF CAPTURED DATA                 -->
            <!--                 OPTIONAL, default to entire pkt              -->
            <!--               can be overriden on a per flow basis           -->
            <!--                                                              -->
            <!-- IMPORTANT NOTE:                                              -->
            <!-- Chunks will be concatenated in the order specified.          -->
            <!-- Overlapping chunks are concatenanted independently,          -->
```

```
            <!-- possibly leading to redundant data segments.                -->
            <!-- Nevertheless, this will give same signature in different    -->
            <!--  probes.                                                    -->

<!ELEMENT RoundTripDelay  EMPTY>
<!ATTLIST RoundTripDelay  protocol     (icmpEcho|udpNTP|udpSNMP|udpDNS
                                        |tcpSynAck|tcpOptRTTM
                                        |tcpDataAck)              #REQUIRED
                          with         (initiator|responder
                                        |both|unspecified)       "unspecified"
>
<!-- Measuring Round-trip Delays for specific protocol exchanges:           -->
<!-- ===========================================================           -->
<!--   The probe itself can match corresponding pairs of data units,       -->
<!--   one per direction, and therefore compute the round-trip time        -->
<!--   between the probing point and one of the communication side.        -->
<!--                                                                       -->
<!--     ==> Measurements can be individual (from probe to collector to    -->
<!--         database), or aggregated in the probe, or individually        -->
<!--         forwarded and aggregated by the collector                     -->
<!--         As specified by <Delay> directive                             -->
<!--                                                                       -->
<!--             target:    Which data unit to measure?                    -->
<!--                          (Default "firstFragment")                    -->
<!--               firstFragment One capture timestamp for single          -->
<!--                             or first IP fragment of a datagram/segment -->
<!--                             ==> IP reassembly is not required for this -->
<!--                                 (second and subsequent fragments are   -->
<!--                                                    ignored for this) -->
<!--               allFragments  Meaningless in case of round-trip          -->
<!--               datagram      One capture timestamp for unfragmented pkt -->
<!--                             But two capture timestamps for chains of   -->
<!--                             datagram fragments: {first seen, last seen}-->
<!--                             ==> IP reassembly is required for this     -->
<!--                             ==> Round-Trip                             -->
<!--                                   = firstTS(ingress) -lastTS(egress)   -->
<!--             protocol:  -What upper layer protocol is covered by the    -->
<!--                         <FlowClass> definition (<Filter> MUST BE      -->
<!--                         PROPERLY EXPRESSED TO MATCH THIS PROTOCOL)    -->
<!--                        -Hence specifies which upper layer protocol    -->
<!--                         analysis has to be applied on <FlowClass>     -->
<!--                         matching packets.                             -->
<!--               TCP connections:                                        -->
<!--                         Identifies which further peer packets matching -->
<!--                         is required.                                  -->
<!--                   Syn/Syn+Ack: 'with' is implied, its given value is not-->
<!--                         considered.                                   -->
<!--               udpSNMP or                                              -->
<!--               udpDNS:   Round-trip between the probing point and the  -->
<!--                         SNMP Agent or DNS Server                      -->
<!--               udpNTP:  -Round-trip between the probing point and the  -->
<!--                         NTP Server, but query/reply Transmit Time     -->
<!--                         permit to know latency between probe and each -->
<!--                         end (when these have there time precisely     -->
<!--                         synchronised); hence:                         -->
<!--                        -ALSO deduced Round-Trip between NTP client and -->
<!--                         NTP server:                                   -->
<!--                             ==> Round-Trip(client, server)            -->
<!--                                   = RTT(probe, server)                -->
<!--                                           + 2 * Delay(client, probe)-->

<!ELEMENT InterPacket     EMPTY >

<!-- Measuring Inter-Packet (inter-data-unit) Delays:                       -->
<!-- ================================================                      -->
<!--   Means that the probe itself can compute the delay between capture   -->
<!--   timestamps of individual packets or reassembled datagram units      -->
```

```
<!--                                                                        -->
<!--     ==> Measurements can be individual (from probe to collector to     -->
<!--         database), or aggregated in the probe, or individually         -->
<!--         forwarded and aggregated by the collector                      -->
<!--         As specified by <Delay> directive                              -->
<!--                                                                        -->
<!--             target:    Which successive data unit to measure?          -->
<!--               firstFragment Compare TS of single                       -->
<!--                             or first IP fragment of a datagram/segment -->
<!--                             ==> IP reassembly is not required for this -->
<!--                                 (second and subsequent fragments are   -->
<!--                                                   ignored for this) -->
<!--               allFragments  Compare every fragment with its successor  -->
<!--               datagram      One capture timestamp for unfragmented pkt -->
<!--                             But two capture timestamps for chains of   -->
<!--                             datagram fragments: {first seen, last seen}-->
<!--                             ==> IP reassembly is required for this     -->
<!--                             ==> InterDatagram delay                    -->
<!--                                     = firstTS(next) - lastTS(previous) -->

<!ELEMENT Histogram       EMPTY >
<!ATTLIST Histogram       lowBound     NMTOKEN                  #REQUIRED
                          highBound    NMTOKEN                  #REQUIRED
                          sliceCount   NMTOKEN                  "12"
>

<!-- For any three types of Delay, EITHER "collectorAggregated" or          -->
<!-- "probeAggregated" Histogram MAY (optional) be specified to preserve    -->
<!-- several ranges of observed delay values:                               -->
<!--                                                                        -->
<!-- When Histogram is not specified, all values are aggregated over time   -->
<!-- period into a single unbound bucket (no histogram slices)              -->
<!--                                                                        -->
<!--               lowBound:  First slice covers all values <  lowBound     -->
<!--              highBound:  Last  slice covers all values >= highBound    -->
<!--             sliceCount:  In between first and last slices, there are   -->
<!--                          sliceCount-2 intervals of values, of equal    -->
<!--                          length                                        -->
<!--       Currently only used for Delay metrics, the values are            -->
<!--       signed integers coded in 32 bit. Specifying any bound            -->
<!--       as either INT32_MIN (0xffff or -2147483648)                      -->
<!--              or INT32_MAX (0x7fff or  2147483647)                      -->
<!--       means "unbound".                                                 -->

<!ELEMENT Stats           (PacketCounters?,TCPConnections?,FileTransfers?) >
<!ATTLIST Stats           verifChksum  (none|bestEffort|fullReassembly)
                                                                  #REQUIRED
>
<!ELEMENT PacketCounters  EMPTY >
<!ATTLIST PacketCounters  for          (firstFragment|allFragments
                                        |datagram)                #REQUIRED
>

<!ELEMENT TCPConnections  EMPTY >
<!ATTLIST TCPConnections  granularity  (each|groupedByCollector
                                        |groupedByProbe)          "each"
>

<!ELEMENT FileTransfers   EMPTY >
<!ATTLIST FileTransfers   protocol     (FTP|HTTP)                 #REQUIRED
                          granularity  (each|groupedByCollector
                                        |groupedByProbe)          "each"
                          ftpdata      (start-stop|full)          #IMPLIED
>
            <!-- Measuring FTP File Transfers                               -->
            <!-- ++++++++++++++++++++++++++++                               -->
            <!-- protocol="FTP" ==> ftpdata field must be specified         -->
```

```
            <!--                    to permit matching the associated FTP   -->
            <!--                    data connections that would not         -->
            <!--                    otherwise be analysed by other Flow     -->
            <!--                    Classes Filters.                        -->
            <!-- ftpdata: start-stop:                                       -->
            <!--                    Only look at SYN (or first seen) packet -->
            <!--                    and at FIN or RST packet of any TCP     -->
            <!--                    connection between the pair of IP       -->
            <!--                    addresses of an FTP Control connection  -->
            <!-- ftpdata: full:                                             -->
            <!--                    Look at every packet  of any TCP        -->
            <!--                    connection between the pair of IP       -->
            <!--                    addresses of an FTP Control connection  -->
            <!-- RESTRICTION : Only works when client IP (PORT) or server   -->
            <!--               IP (PASV) stays the same between Control and -->
            <!--               Data connections.                            -->


<!ELEMENT FlowGrain        EMPTY >
<!ATTLIST FlowGrain        ref          IDREF                      #REQUIRED
>

<!ELEMENT Filter           (On+, CaptureTimeStamp?,
                           (IpOptTimestamp | NtpTimestamp)*, NatPat*,
                                        PacketExpr, FlowGrain?)>

<!ELEMENT On               EMPTY >
<!ATTLIST On               probe        IDREF                      #REQUIRED
                           if           NMTOKEN                    #REQUIRED
>
            <!-- probe:        a reference to the name of the probe         -->
            <!--               to which the Packet Filter Expression applies-->
            <!-- if:           a reference to the probe interface name      -->
            <!--               to which the Packet Filter Expression applies-->

<!ELEMENT CaptureTimeStamp EMPTY >
<!ATTLIST CaptureTimeStamp
                           hopName      NMTOKEN                    #REQUIRED
>
<!ELEMENT IpOptTimestamp   EMPTY >
<!ATTLIST IpOptTimestamp   ipTSNum      NMTOKEN                    #REQUIRED
                           hopName      NMTOKEN                    #REQUIRED
>

<!ELEMENT NtpTimestamp     EMPTY >
<!ATTLIST NtpTimestamp     ntpTime      (originreq|recvdreq
                                        |xmitreq|originrsp
                                        |recvdrsp|xmitrsp)         #REQUIRED
                           hopName      NMTOKEN                    #REQUIRED
>
            <!-- Capture Timestamps on packet/datagram                    -->
            <!-- ================================================        -->
            <!--   name of the hop timestamp at a probing point          -->
            <!--                                                          -->
            <!-- Optional Additional Timestamps on packet/datagram        -->
            <!-- ================================================        -->
            <!--                                                          -->
            <!-- + based on IP OPTION TIMESTAMP (milliseconds since midnight-->
            <!--     The nth (ipTSNum) IP timestamp in the flow packets is  -->
            <!--     mapped to the given hopName                          -->
            <!--     1 <= ipTSNum <= 9                                    -->
            <!--                                                          -->
            <!-- + based on UDP NTP Embedded Timestamps                   -->
            <!--     Designates, respectively for a request or response,  -->
            <!--     which of the three relevant NTP Timestamps to report -->
            <!--     as hopName                                           -->
            <!--     This requires firstFragment|datagram                 -->
```

```
<!ELEMENT NatPat          EMPTY>
<!ATTLIST NatPat          translate    (src|dst|addr
                                        |sport|dport|port)        #REQUIRED
                          from         CDATA                     #IMPLIED
                          into         CDATA                     #REQUIRED
>
            <!-- Address and/or Port Translation for Probing Point(s)       -->
            <!-- ================================================       -->
            <!--                                                           -->
            <!-- Granular flows corresponding to this FlowClass and        -->
            <!--  discovered at the given probing point(s) (as per <On>    -->
            <!--  clause(s)> must translate their identification information-->
            <!--  transmitted to the collector according to the given      -->
            <!--  <NatPat> directive(s) for permitting common network-wide -->
            <!--  determination of same flows independent on their local   -->
            <!--  translation on IP address(es) and UDP/TCP port number(s) -->
            <!--                                                           -->
            <!-- translate:                                                -->
            <!--  src or dst: specifies that the source or destination     -->
            <!--              locally seen IP address is subject to        -->
            <!--              translation before being mentioned to the    -->
            <!--              collector.                                   -->
            <!--  addr:       specifies that both the source and destination-->
            <!--              locally seen IP addresses are subject to     -->
            <!--              translation before being mentioned to the    -->
            <!--              collector.                                   -->
            <!--              ==> only meaningful with explicit 'from' value-->
            <!--  sport or dport:                                          -->
            <!--              specifies that the source port number or     -->
            <!--              destination port number locally seen value has-->
            <!--              to be translated before being mentioned to the-->
            <!--              collector.                                   -->
            <!--  port:       specifies that both the source and destination-->
            <!--              locally seen port number values are subject to-->
            <!--              translation before being mentioned to the    -->
            <!--              collector.                                   -->
            <!--              ==> only meaningful with explicit 'from' value-->
            <!-- from   is OPTIONAL                                        -->
            <!--   absent:   replace the value of the field indicated by   -->
            <!--             'translate' whatever it is, with the value    -->             <!--
provided by 'into'                          -->
            <!--            ==> only 'src', 'dst, 'sport' or dport' are    -->
            <!--                meaningful here),                          -->
            <!--   present:  gives the value of the field(s) indicated by  -->
            <!--             'translate' which would be subject to be      -->
            <!--             replaced by that given by 'into'              -->
            <!-- into:  the replacing value for field(s) designated by     -->
            <!--        'translate'                                        -->


<!ELEMENT PacketExpr      (Predicate|AND|NotAND)>
<!ELEMENT AND             (Predicate|OR|NotOR)+>
<!ELEMENT NotAND          (Predicate|OR|NotOR)+>
<!ELEMENT OR              (Predicate|SubAND|SubNotAND)+>
<!ELEMENT NotOR           (Predicate|SubAND|SubNotAND)+>
<!ELEMENT SubAND          EMPTY>
<!ELEMENT SubNotAND       EMPTY>
            <!-- Packet Filter Expression for <FlowClass> match            -->
            <!-- ==============================================            -->
            <!-- This expression has up to three layers of sub-expressions. -->
            <!-- Either Single-predicate:                                  -->
            <!--     <PacketExpr>                                          -->
            <!--         <Predicate field=... op=... value=.../>           -->
            <!--     </PacketExpr>                                         -->
            <!-- Or predicate AND predicate AND predicate ...              -->
```

```
            <!--      <PacketExpr>                                          -->
            <!--         <AND>                                              -->
            <!--            <Predicate field=... op=... value=.../>         -->
            <!--            <Predicate field=... op=... value=.../>         -->
            <!--            ...                                             -->
            <!--         </AND>                                             -->
            <!--     </PacketExpr>                                          -->
            <!-- Or NOT (predicate AND predicate AND predicate ...)         -->
            <!--      <PacketExpr>                                          -->
            <!--         <NotAND>                                           -->
            <!--            <Predicate field=... op=... value=.../>         -->
            <!--            <Predicate field=... op=... value=.../>         -->
            <!--            ...                                             -->
            <!--         </NotAND>                                          -->
            <!--     </PacketExpr>                                          -->
            <!--                                                            -->
            <!-- The list inside <AND> ... </AND> can also contain 1 or more-->
            <!-- alternatives of predicates, introducing a second level     -->
            <!-- with the <OR> ... </OR> or its contrary <NotOR> .. </NotOR>-->
            <!--                                                            -->
            <!-- For sake of completeness, elements joined by an <OR> or    -->
            <!-- <NotOR> connectives can be a mix of predicates and of      -->
            <!-- sub-expressions consisting only in predicates, assembled by-->
            <!-- logical AND connective (<SubAND> tag) or their contrary    -->
            <!-- (<SubNorAND> tag)                                          -->

<!ELEMENT Predicate       EMPTY>
<!ATTLIST Predicate       field        (src|dst|addr
                                        |ipsize|ttl|tosprec|tosdscp|tosbyte
                                        |df|mf|fragofst
                                        |proto
                                        |icmpTypeCode
                                        |sport|dport|port
                                        |syn|fin|ack|psh|urg)     #REQUIRED
                          op           (eq|ne|lt|le|gt|ge|betw|mask) #REQUIRED
                          value        CDATA                      #REQUIRED
                          value2       CDATA                      #IMPLIED
>
            <!-- src or dst: Predicate applies respectively to the source   -->
            <!--             or destination IP address field of the captured-->
            <!--             packet                                         -->
            <!-- addr:       Predicate applies any of source or destination -->
            <!--             IP address fields of the captured packet       -->
            <!--      The 'mask' operator permits to test subnet membership -->
            <!-- ipsize:     Length of the IP fragment, in bytes, including -->
            <!--             the 20 bytes of IPv4 header and the potential  -->
            <!--             IPv4 options data.                             -->
            <!-- ttl:        The IP Time-to-Leave byte value                -->
            <!-- tosprec:    The numeric prority (precedence) given in the  -->
            <!--             three high-order bits of the Type-of-Service   -->
            <!--             byte.                                          -->
            <!-- tosdscp:    The numeric Differentiated Service Code Point  -->
            <!--             given in the six high-order bits of the IPv4   -->
            <!--             Type-of-Service byte.                          -->
            <!-- tosbyte:    The numeric value of the complete IPv4         -->
            <!--             Type-of-Service byte.                          -->
            <!--      Subject to 'mask' operator                            -->
            <!-- df:         The boolean "Don't Fragment" IPv4 flag bit     -->
            <!-- mf:         The boolean "More Fragment" IPv4 flag bit      -->
            <!-- fragofst:   The value of the IPv4 Fragment Offset,         -->
            <!--             expressed in 8-byte words                      -->
            <!--    Unfragmented pkt: (mf == 0) AND (fragofst == 0)         -->
            <!--    First fragment or unfragmented pkt: (fragofst == 0)     -->
            <!-- proto:      valid operators are "eq", "ne", "mask"         -->
            <!--             valid value are "esp", "ah", "udp", "tcp",     -->
            <!--             "icmp", "icmpEcho", "icmpFragNeed",            -->
            <!--             "icmpSrcQuench", "icmpTTLexpired",             -->
```

```
            <!--              "icmpReassemblyFailed", "icmpUnreachable",     -->
            <!--              "icmpOtherError", "icmpInfo", "icmpOther"      -->
            <!--    Note that upper layer protocols are not know at          -->
            <!--         <PacketExpr> level, applied on individual packet.   -->
            <!--         This <FlowFilter> is precisely used for determining-->
            <!--         <FlowClass> membership, whose <Measure> statements  -->
            <!--         permit to identify applicable upper layer protocol  -->
            <!--         analysis. Hence this filter permits the user to     -->
            <!--         specify the upper layer protocol, (typically based  -->
            <!--         TCP/UDP service 'port').                             -->
            <!-- icmpTypeCode is an alternative to 'proto'. It designates    -->
            <!--              the 2-byte ICMP Type and ICMP code of an ICMP  -->
            <!--              packet.                                        -->
            <!--      Subject to 'mask' operator                             -->

<!ELEMENT Condition       (StatePred|AND_list|NotAND_list|OR_list|NotOR_list)>
<!ELEMENT AND_list        (StatePred+)>
<!ELEMENT NotAND_list     (StatePred+)>
<!ELEMENT OR_list         (StatePred+)>
<!ELEMENT NotOR_list      (StatePred+)>

<!ELEMENT StatePred       EMPTY>
<!ATTLIST StatePred       state        (fragnum|fragcnt
                                        |dgramsize)              #REQUIRED
                          op           (eq|ne|lt|le|gt|ge|betw)  #REQUIRED
                          value        CDATA                     #REQUIRED
                          value2       CDATA                     #IMPLIED
>
                               <!-- Not yet implemented                    -->
                               <!--     |tcpwindow                         -->
                               <!--     |tcpretransmitted                  -->
                               <!--     |tcpdirtyopen                      -->
                               <!--     |tcpdirtyclose                     -->
                               <!--     |tcpemptyconn                      -->
                               <!--     |ftpemptysession                   -->
                               <!--     |ftpnofiletransfer                 -->
                               <!--     |ftpactivetransfer                 -->
                               <!--     |ftppassivetransfer)      #REQUIRED -->
            <!-- The <FlowClass> optional <Condition> permits to express    -->
            <!-- additional contraints on matching packets, based on result -->
            <!-- of statefull analysis (e.g. IP reassembly, or TCP          -->
            <!-- connection follow-on)                                      -->
            <!--                                                           -->
            <!-- Only a single-level of predicates can be expressed,       -->
            <!-- assembled by one of the following boolean connectives:    -->
            <!--  none:           single state predicate only              -->
            <!--  <AND_list>:     all state predicates must be true        -->
            <!--  <NotAND_list>:  at least one of the predicates is false  -->
            <!--  <OR_list>:      at least one of the predicates is true   -->
            <!--  <NotOR_list>:   all state predicates must be false       -->
            <!--                                                           -->
            <!-- fragnum:    IP Fragment number (first is 1)               -->
            <!-- fragcnt:    Number of IP fragment in datagram/segment     -->
            <!--             (==1 for single fragment)                     -->
            <!-- dgramsize:  How many bytes of UDP payload, TCP payload or -->
            <!--             otherwise IP payload, after IPv4 reassembly   -->
            <!--             (single frag. or reassembled datagram/segment) -->
            <!-- tcpwindow:  Size in bytes of the TCP window, after agreed -->
            <!--             TCP window scale option                       -->
            <!-- tcpretransmitted:                                         -->
            <!--             Whether (==true) or not (==false)             -->
            <!--             the TCP segment contains retransmitted payload -->
            <!--             data                                          -->
            <!-- tcpdirtyopen:                                             -->
            <!--             The corresponding TCP connection has been      -->
            <!--             initiated with one SYN or a SYN/SYN+ACK       -->
            <!--             but has not been continued (no complete 3-way -->
```

```
            <!--              SYN/SYN+ACK/ACK handshake seen during timeout  -->
            <!--              period                                         -->
            <!-- tcpdirtyclose:                                              -->
            <!--              The corresponding TCP connection has been      -->
            <!--              closed, either, by RESET or single FIN packet  -->
            <!--              without reverse FIN packet seen during timeout -->
            <!--              period                                         -->
            <!-- tcpemptyconn:                                               -->
            <!--              Not any payload data byte has been passed,     -->
            <!--              either way, over the connection                -->
            <!-- ftpemptysession:                                            -->
            <!--              Not any useful FTP command has been passed,    -->
            <!--              over the control connection                    -->
            <!-- ftpnofiletransfer:                                          -->
            <!--              The FTP control connection didn't initiate any -->
            <!--              actual file transfer (e.g. poll only)          -->
            <!-- ftpactivetransfer:                                          -->
            <!--              The FTP control connection or the resulting    -->
            <!--              FTP data connection involves active FTP mode   -->
            <!--              (i.e. PORT command)                            -->
            <!-- ftppassivetransfer:                                         -->
            <!--              The FTP control connection or the resulting    -->
            <!--              FTP data connection involves passive FTP mode  -->
            <!--              (i.e. PASV command)                            -->
```

6.1.2 Sample XML Configuration of Single Probe with Two Interfaces

```
<!-- trafMon sample configuration file for Single Probe -->
<!-- @(#) singleProbe_sample.xml $Id: 202399b85754147d6bc7600ac6a5fd35a886508f $
-->

<!DOCTYPE trafMonConfig SYSTEM "tmon.dtd" [

<!-- trafMon Systems -->
<!ENTITY trafMon-svr  "172.16.12.34">
<!ENTITY trafMon-prb  "172.20.26.63">
<!-- PDU port (for sending PDUs from trafMon probe to trafMon server) -->
<!ENTITY tmonProbe "9877">
<!ENTITY tmonColl  "9878">
<!-- Temporary storage for CSV data produced by Central Processor -->
<!ENTITY dataPath "/var/trafMon/collector">
<!-- Service Port numbers -->
<!ENTITY dns     "53">
<!ENTITY ntp    "123">
<!ENTITY iperf "5001">
<!ENTITY http    "80">
<!ENTITY snmp   "161">
<!ENTITY ftp     "21">
<!ENTITY ftpdata "20">
<!-- End of ENTITIES -->
]>

<trafMonConfig serial="2" startAt="2016-08-26 12:00:00" pktSignBytes="3"
  maxTravelTime="30000" >
```

```
<Collector name="trafMon-server" ID="100"
  descr="tmon_collector on MyNetwork at NOC center"
  burstRate="1000">
  <Addr ip="&trafMon-svr;" port="&tmonColl;" UDPBufferSize="20000"/>
  <Output dataFile="&dataPath;/observations.%y%m%d%H%M"
          eventFile="&dataPath;/events.%y%m%d%H%M"
          excepFile="&dataPath;/exceptions.%y%m%d%H%M"
          period="5"  />
</Collector>

<Probe name="trafMon-probe" ID="11" descr="single trafMon Probe">
  <Interface name="p1p1" ID="111" descr="DMZ capture" snapLen="210"
    bufPacketCount="10000"
    expr= "ip"
    />
  <Interface name="p1p2" ID="112" descr="INTERAL capture" snapLen="210"
    bufPacketCount="10000"
    expr= "ip"
    />
  <PDUSending probePort="&tmonProbe;">
    <SendTo collector="trafMon-server" maxPDUSize="1460" minTimeGap="1"
      maxPDUBuildTime="5" heartBeatDelay="90" timeout="20" retries="5"
      TOMult="1" TOIncr="5"  breakBorderTime="180" dropObsFinalTimeout="5" />
  </PDUSending>
</Probe>

<!-- ============================                                    -->

<GranularFlow name="protoConversAtProbeIf" >
  <DistinctIf /> <!-- mandatory when Counters, to avoid double records -->
  <DistinctAddr field="addrpair" />
  <DistinctPort field="portpair" portspec="privileged" />
  <GroupBy      field="ipproto"/>
</GranularFlow>

<GranularFlow name="protoConversAtProbeIf_HighPort" >
  <DistinctIf /> <!-- mandatory when Counters, to avoid double records -->
  <DistinctAddr field="addrpair" />
  <DistinctPort field="portpair" portspec="alldistinct" />
                <!-- When service protocol port may be >= 1024 -->
  <GroupBy      field="ipproto"/>
</GranularFlow>

<GranularFlow name="uniDirAtProbeIf" >
  <DistinctIf /> <!-- mandatory when Counters, to avoid double records -->
  <DistinctAddr field="srcdst" />
  <DistinctPort field="portpair" portspec="privileged" />
  <GroupBy      field="ipproto"/>
</GranularFlow>


<!-- FTP: TCP port 21
     ================
-->
```

```
<FlowClass id="21" name="FTP_port21" descr="TCP with port==21">
  <Measure interval="1min" >
    <Stats verifChksum="bestEffort">
      <PacketCounters for="firstFragment"/>
                      <!-- Don't ask for Dgram for TCP to avoid unnecessary
                           keeping of subsequent frags (of other flows)
                           between same IP address pair -->
      <TCPConnections granularity="each"/>
      <FileTransfers  protocol="FTP" granularity="each"
                      ftpdata="full"/>
    </Stats>
  </Measure>
  <FlowGrain ref="protoConversAtProbeIf" />
  <Filter>
    <On probe="trafMon-probe" if="p1p1" />
    <On probe="trafMon-probe" if="p1p2" />
    <PacketExpr>
      <AND>
        <Predicate field="proto" op="eq" value="tcp"/>
        <Predicate field="port"  op="eq" value="21"/>
      </AND>
    </PacketExpr>
  </Filter>
</FlowClass>

<!-- HTTP: TCP port 80, 443, 8080 or 8443
     ====================================
-->
<FlowClass id="80" name="HTTP" descr="TCP with port==[80,443,8080,8443">
  <Measure interval="1min" >
    <Stats verifChksum="bestEffort">
      <PacketCounters for="firstFragment"/>
                      <!-- Don't ask for Dgram for TCP to avoid unnecessary
                           keeping of subsequent frags (of other flows)
                           between same IP address pair -->
      <TCPConnections granularity="each"/>
    </Stats>
  </Measure>
  <FlowGrain ref="protoConversAtProbeIf_HighPort" />
  <Filter>
    <On probe="trafMon-probe" if="p1p1" />
    <On probe="trafMon-probe" if="p1p2" />
    <PacketExpr>
      <AND>
        <Predicate field="proto" op="eq" value="tcp"/>
        <OR>
          <Predicate field="port"  op="eq" value="80"/>
          <Predicate field="port"  op="eq" value="443"/>
          <Predicate field="port"  op="eq" value="8080"/>
          <Predicate field="port"  op="eq" value="8443"/>
        </OR>
      </AND>
    </PacketExpr>
  </Filter>
```

```
</FlowClass>

<!-- ALL Unidirectional packets (for volumes counting)
     ================================================
-->
<FlowClass id="200" name="ALL_packets"
                                        descr="ALL Unidirectional IP Fragments">
  <Measure interval="1min" >
    <Stats verifChksum="bestEffort">
      <PacketCounters for="allFragments"/>
    </Stats>
  </Measure>
  <FlowGrain ref="uniDirAtProbeIf" />
  <Filter>
    <On probe="trafMon-probe" if="p1p1" />
    <On probe="trafMon-probe" if="p1p2" />
    <PacketExpr>
      <AND>
          <Predicate field="src" op="betw"
                                  value="0.0.0.1" value2="255.255.255.254" />
          <Predicate field="dst" op="betw"
                                  value="0.0.0.1" value2="255.255.255.254" />
      </AND>
    </PacketExpr>
  </Filter>
</FlowClass>

<!-- Round-Trip Delays
     =================
-->
<!--  Round trip delay measurement for ICMP from INTERNAL LAN
-->
<FlowClass id="1111" name="Echo-RoundTrip-histo"
  descr="ICMP Echo Req/Rsp aggregated delay">
  <Measure interval="1min" >
    <Delay for="firstFragment" granularity="probeAggregated">
      <RoundTripDelay protocol="icmpEcho" />
      <Histogram lowBound="0" highBound="5000" sliceCount="5" />
    </Delay>
  </Measure>
  <FlowGrain ref="protoConversAtProbeIf" />
  <Filter>
    <On probe="trafMon-probe" if="p1p2" />
    <PacketExpr>
      <AND>
        <Predicate field="proto" op="eq" value="icmpEcho"/>
      </AND>
    </PacketExpr>
  </Filter>
</FlowClass>
<!-- Round trip delay measurement for DNS
-->
<FlowClass id="2222" name="DNS-RoundTrip-histo"
  descr="DNS Req/Rsp aggregated delay">
```

```
    <Measure interval="1min" >
      <Delay for="firstFragment" granularity="probeAggregated">
        <RoundTripDelay protocol="udpDNS" />
        <Histogram lowBound="0" highBound="800" sliceCount="4" />
      </Delay>
    </Measure>
    <FlowGrain ref="protoConversAtProbeIf" />
    <Filter>
      <On probe="trafMon-probe" if="p1p1" />
      <On probe="trafMon-probe" if="p1p2" />
      <PacketExpr>
        <AND>
          <Predicate field="proto" op="eq" value="udp"/>
          <Predicate field="port"  op="eq" value="53"/>
        </AND>
      </PacketExpr>
    </Filter>
  </FlowClass>
  <!-- Round trip delay measurement for SNMP from INTERNAL LAN
  -->
  <FlowClass id="3333" name="SNMP-RoundTrip-histo"
    descr="SNMP Req/Rsp aggregated delay">
    <Measure interval="1min" >
      <Delay for="firstFragment" granularity="probeAggregated">
        <RoundTripDelay protocol="udpSNMP" />
        <Histogram lowBound="0" highBound="800" sliceCount="4" />
      </Delay>
    </Measure>
    <FlowGrain ref="protoConversAtProbeIf" />
    <Filter>
      <On probe="trafMon-probe" if="p1p2" />
      <PacketExpr>
        <AND>
          <Predicate field="proto" op="eq" value="udp"/>
          <Predicate field="port"  op="eq" value="161"/>
        </AND>
      </PacketExpr>
    </Filter>
  </FlowClass>
  <!-- Round trip delay measurement for NTP
  -->
  <FlowClass id="4444" name="NTP-RoundTrip-histo"
    descr="NTP Req/Rsp aggregated delay">
    <Measure interval="1min" >
      <Delay for="firstFragment" granularity="probeAggregated">
        <RoundTripDelay protocol="udpNTP" with="both" />
        <Histogram lowBound="0" highBound="1000" sliceCount="5" />
      </Delay>
    </Measure>
    <FlowGrain ref="protoConversAtProbeIf" />
    <Filter>
      <On probe="trafMon-probe" if="p1p1" />
      <On probe="trafMon-probe" if="p1p2" />
      <PacketExpr>
```

```
        <AND>
          <Predicate field="proto" op="eq" value="udp"/>
          <Predicate field="port" op="eq" value="123"/>
        </AND>
      </PacketExpr>
    </Filter>
  </FlowClass>
  <!-- Round trip delay measurement for TCP-SYN
  -->
  <FlowClass id="5555" name="TCP-SYN-RoundTrip-histo"
    descr="TCP SYN/SYN+ACK aggregated delay">
    <Measure interval="1min" >
      <Delay for="firstFragment" granularity="probeAggregated">
        <RoundTripDelay protocol="tcpSynAck" />
        <Histogram lowBound="0" highBound="2000" sliceCount="6" />
      </Delay>
    </Measure>
    <FlowGrain ref="protoConversAtProbeIf" />
    <Filter>
      <On probe="trafMon-probe" if="p1p1" />
      <On probe="trafMon-probe" if="p1p2" />
      <PacketExpr>
        <AND>
          <Predicate field="proto" op="eq" value="tcp"/>
        </AND>
      </PacketExpr>
    </Filter>
  </FlowClass>
  <!-- Round trip delay measurement for TCP-RTTM -->
  <!-- NOT RELIABLE: Linux TCP sends a bunch of pkts with same timestamps
                     and succesive corresponding ACKs come back each with
                     increased delay (increased queuing time in peer entity
  <FlowClass id="6666" name="TCP-RTTM-RoundTrip-histo"
    descr="TCP RTTM Timestamps aggregated delay">
    <Measure interval="1min" >
      <Delay for="firstFragment" granularity="probeAggregated">
        <RoundTripDelay protocol="tcpOptRTTM" with="both" />
        <Histogram lowBound="0" highBound="300" sliceCount="8" />
      </Delay>
    </Measure>
    <FlowGrain ref="protoConversAtProbeIf" />
    <Filter>
      <On probe="trafMon-probe" if="p1p1" />
      <On probe="trafMon-probe" if="p1p1" />
      <PacketExpr>
        <AND>
          <Predicate field="proto" op="eq" value="tcp"/>
        </AND>
      </PacketExpr>
    </Filter>
  </FlowClass>
  -->
</trafMonConfig>
```