NetMate

User and Developer Manual

February 12, 2004

Editor: Carsten Schmoll

Authors: Carsten Schmoll, Sebastian Zander

Table of Contents

1 Introduction

This document is the user and developer manual of the NetMate software suite.

Chapter 2 provides an overview about the software and explains the terminology used.

Chapter 3 contains information for all users. It describes how to install and use the software and contains numerous examples.

Chapter 4 contains information for developers. It explains how to write new packet processing and export modules. It also explains how to write new classifiers, packet samplers and how to extend the core meter.

Chapter 5 lists the references.

Chapter 6 is the appendix and contains the definitions of various file formats as well as the control and export protocol(s).

2 Overview

The name of the project is NetMate and the software meter is called "**netmate**". This is a creative abbreviation for: **NET**work **M**easurement and **A**ccoun**T**ing syst**E**m

Additional tools in this software toolkit are:

nmctl netmate control: a shell script to control the start, restart and termination of netmate as a background process; can also query the running status

nmrsh netmate remote shell: a UNIX shell-like program to interactively configure and query the Meter at run-time; can also send batched commands

The following terminology is used within this document:

Term	Description
Classification	Process of identifying which rules match an observed packet
Export Module	Meter module which exports flow records
Flow Record	Metric Data for a particular flow generated at a certain time
Measurement Policy	Other term for Rule(s) or Ruleset
Meter Data	Flow Record(s)
Meter Manager	Software for (remote) control of the meter
Meter Task	Task running on a meter. This can be for example a running Rule or a get information/status request etc.
Meter	Measurement software
Metric	Defines what to measure e.g. packet counter, loss, delay etc.
Observation Point	Point in the network where the meter is attached and observes packets
Processing Module	Meter module which computes metric data based on observed and classified packets
Rule	Describes the conditions of a measurement e.g. the flow selection criteria, the type of measurement, the measurement data export etc.
Ruleset	Number of rules with a potential association among them (e.g. logical grouping).

NetMate is a network traffic meter and monitoring tool. Its job is to listen to network traffic, to classify packets into flows and to compute metrics for flows. The overall system architecture is depicted in the next figure. The system architecture basically follows the principle of the RTFM [rtfm] architecture with some slight changes. Netmate is the meter which listens on a network interface, classifies packets and computes metrics. The operations can be (remotely) controlled by either the nmrsh tool or a standard web browser e.g. Netscape, Mozilla or Internet Explorer. The computed metric data is delivered to a meter data collector. Currently netmate does not support the export of data via the network to a remote host, and therefore no meter data collector exists. Export results are only stored locally. Exporting data via the IPFIX [ipfix] protocol is under development and in the next release the NetMate project will contain an IPFIX collector. The analysis application is currently considered out of scope and is shown for convenience only. Work on a graphical analysis tool for netmate results has just been started.

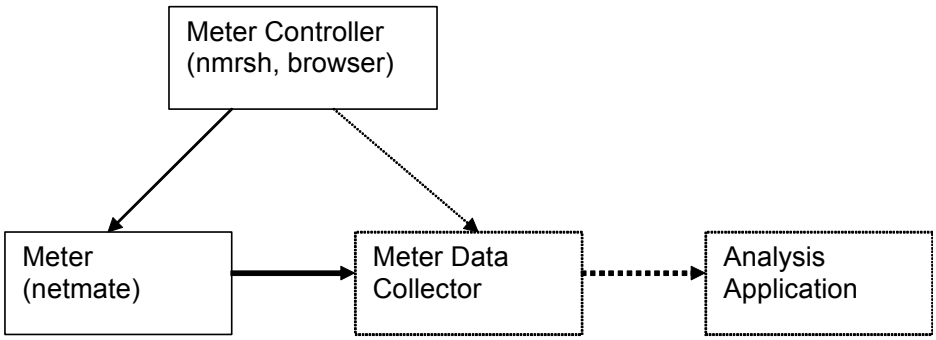

Figure 2-1: Architecture Overview

For more information please read the design document.

3 Using NetMate

3.1 Installation

3.1.1 RPM Package

Binary packages adhere to the following naming convention:

```
netmate-<version>.tar.gz
```

1. Download the rpm package for your hardware and OS from the netmate web page
2. Install the package (you will probably need root permissions to do this)

```
rpm –ihv <package-name>
```

When updating from an older version you will need to run rpm with:

```
rpm –Uhv <package-name>
```

It is also possible to download a source rpm or tarball package from the netmate website in case you want to build your own customized version.

In order to run netmate successfully you will need certain libraries installed on your system. The list of required libraries is also part of the next manual section.

3.1.2 Source Package

Download the source package from the web and unpack the sources:

```
tar xvzf <netmate.tar.gz>
```

or

```
gunzip <netmate.tar.gz> ; tar xvf <netmeta.tar>
```

(if your tar program does not directly support gzip decompression).

Alternatively, download the latest source RPM file and install with `rpm –ihv <package>`

The netmate toolset has been successfully compiled and run on ix86/Linux 2.4, ix86/FreeBSD 4 and Sparc/Solaris 8.

The following libraries are required:

– readline (http://cnswww.cns.cwru.edu/~chet/readline/rltop.html)

– libpcap (http://www.tcpdump.org)

– libxml2 and libxslt (http://xmlsoft.org)

– libcurl (http://curl.haxx.se)

The following libraries may be required depending on the build configuration:

- openssl (http://www.openssl.org)
- pthreads (OS dependent)

Notes:

- When building the netmate suite from sources you will also need to have the corresponding "XYZ-devel" packages installed on your system (header files).
- libxslt and libcurl are only required for the interactive remote shell meter control tool (nmrsh) and not for the meter itself.
- The process of obtaining and installing the library packages from source or binary packages is out of the scope of this document.

After unpacking the source archive the installation follows the usual steps:

- `./configure [<options>]`
- `make`
- `make install`

You probably need to be root when doing the last command.

To get a full listing of all available configure options invoke it like:

- `./configure --help`

If you want to install netmate in a different file hierarchy than /usr/local, you need to specify that directory when running configure:

- `./configure --prefix=<install_path>`

If you happen to have write permission in that directory, you can do 'make install' without being root. An example of this would be to make a local install in your own home directory:

- `./configure --prefix=$HOME`
- `make`
- `make install`

By default netmate will be compiled without SSL and thread support. SSL can be enabled by:

- `./configure --enable-ssl`

If you have OpenSSL installed in the default search path for your compiler/linker, you don't need to do anything special. If you have OpenSSL installed somewhere else (for example in /opt/OpenSSL,) you can run configure like this:

- `./configure --with-ssl=/opt/OpenSSL`

Thread support requires a functional pthread library and can be enabled with:

- `./configure --enable-threads`

If you have OpenSSL installed, but with the libraries in one place and the header files somewhere else, you have to set the LDFLAGS and CPPFLAGS environment variables prior to running configure. Something like this should work:

- with the Bourne shell and its clones:

```
export CPPFLAGS="-I/path/to/ssl/include" LDFLAGS="-L/path/to/ssl/lib" \
       ./configure
```

- with csh, tcsh and their clones:

```
env CPPFLAGS="-I/path/to/ssl/include" LDFLAGS="-L/path/to/ssl/lib" \
       ./configure
```

If your SSL library was compiled with rsaref (usually for use in the United States), you may also need to set:

```
LIBS=-lRSAglue -lrsaref
```

If you want to compile netmate for debugging use:

- `./configure --enable-debug`

For using SSL a self signed certificate is needed. The certificate will be created during the installation process. You will be asked to enter information which will be incorporated into the certificate such as name, address, country. It is sufficient to just use the default information (hit enter repeatedly).

3.2 Using NetMate

3.2.1 Environment

netmate does not use any specific environment variables. As usual LD_LIBRARY_PATH must include all directories with shared libraries needed by netmate. If netmate fails to run because it cannot find a shared library, locate the library and add the path

(with the Bourne shell and its clones):

```
export LD_LIBRARY_PATH=”$LD_LIBRARY_PATH:/new/path”
```

(with csh, tcsh and their clones):

```
setenv LD_LIBRARY_PATH ”$LD_LIBRARY_PATH:/new/path”
```

nmrsh requires the HOME variable to contain the path to the user's home directory e.g. `HOME=/home/user`. This variable is used to read and write the command history file used by nmrsh which is stored in the home directory.

3.2.2 Running the System – a Quick Start

After the successful installation of the toolkit you may run netmate (or nmrsh, nmctl) as any other command line tool from the shell prompt via:

```
– cd <install_path>/bin ; ./netmate
```

or

```
– <install_path>/bin/netmate
```

For convenience you may add the netmate bin directory to your PATH environment variable (analogous to the description in 3.2.1) or alternatively generate a soft in the form of link from:

```
– ln –s <install_path>/bin/netmate /usr/local/bin/
```

You will probably require root permissions to generate such a soft link. Given that the netmate bin directory is part of your PATH environment variable you now can start it via:

```
– netmate [<options>]
```

The same techniques apply to starting the nmrsh tool and the nmctl script.

When started without any parameters netmate will start up but run without any installed metering rules. This means that no traffic analysis is performed and no output is generated.

There are three ways to add analysis rules to netmate:

1. Start netmate with a ruleset parameter (-r option). This will read rules from a rule file on startup and install them. See section 3.2.3.1 for details.

2. Configure (add/remove) new tasks on-line after startup using the nmrsh tool. This allows adding or removing analysis rules to/from netmate at runtime from a remote host. See section 3.2.4.2 for an example session.

3. Configure (add/remove) new tasks on-line after startup using a web browser. With a standard web browser it is possible to connect to the built-in web server of netmate and add or delete rules on-line from a remote host. See section 3.2.4.3 for details.

All of these three methods can be used in combination.

The typical command line for starting netmate is:

```
– netmate –c <configfile> -r <rulefile>
```

If started successfully this prints out something like:

```
NetMate version 0.6.1, (c) 2003 - 2004 Fraunhofer Institute FOKUS, Germany
netmate logfile is: /var/log/netmate.log
Up and running.
```

Netmate reads its settings from the supplied configuration file. If no file is supplied then the default file <install_path>/etc/netmate/netmate.conf.xml is used. This file contains attribute-value pairs which alter the behaviour of the software. When building your own configuration

file it is encouraged to copy and modify the default netmate.conf.xml. The complete configuration file description can be found in section 3.2.5.2.

The rulefile parameter tells netmate what analysis rule(s) it shall install and run right from the start. The rules describe packet filters and metrics to apply to traffic flows as well as define how to export the metric data. The syntax description for ruleset files can be found in section 3.2.6. Some example rule files to start with are installed in <install_path>/etc/netmate. These files are named "example_rules<num>.xml". Copy and modify these files when starting to experiment with ruleset files.

Netmate has very sparse output to the terminal. Its progress is written to the log file /var/run/netmate.log. This file is best viewed with tools such as less or with tail –f <logfile> which constantly updates the output when new logging messages appear. If required the log file location can be changed via the command line (-l parameter) or via the configuration file.

The next section describes the command line options in detail.

3.2.3 Command Line Arguments

The command line options of the tools in the netmate tool set adhere to the standard UNIX semantics. This means that options are either specified with -x <optionparam> or with --longoptionname <optionparam>. The list of options of a tool is shown with -h or --help.

3.2.3.1 netmate

The following table gives the full list of netmate's command line options:

Short	Long Option	Parameter	Description
-c	--configfile	<file>	Use alternative configuration file instead of default file at <install_path>/etc/netmate/netmate.conf.xml
-C	--fcontfile	<file>	Use alternative file for filter constants
-D	--fdeffile	<file>	Use alternative file for filter definitions
-f	--tracefile	<file>	Use capture file to read packets from instead of live capture from network interface
-h	--help		Show this help and exit
-i	--interface	<interface>	Select network interface to capture from. Default is the first interface, usually "eth0"
-l	--logfile	<file>	Use alternative log file instead of the default file /var/log/netmate.log
-p	--nopromisc		Do not put interface in promiscuous mode, i.e. only capture traffic directly targeted at the machine running netmate.
-P	--fcontfile	<portnum>	Use alternative control port. This port is used to

Short	Long Option	Parameter	Description
			connect to netmate from the nmrsh tool.
-r	--rulefile	<file>	Load tasks from specified rule file on startup
-s	--snapsize	<size>	Specify packet snap size. Smaller size does reduce the CPU load by the meter. Some packet analysis functions however may not work if the size is too small. The default snapsize is stored in the config file
-V	--version		Show version info and exit
-x	--usessl		Use SSL (secure sockets layer) for control communication.

Table 3-1: netmate command line arguments

3.2.3.2 nmrsh

The following table gives the full list of nmrsh's command line options:

Short	Long Option	Parameter	Description
-c	--command	<command>	Command which will be executed, see section 3.2.5.1 for a comprehensive description of the possible commands and their syntax
-f	--cmdfile	<filename>	Select command file name; read commands from a file and execute them in order (non interactive mode)
-h	--help		Show this help and exit
-p	--port	<portnum>	Select netmate port to connect to
-P	--password	<password>	Specify password to use for connection. Typing the connection password in interactive mode is encouraged over using the –P option because the password may appear in ps and similar commands. –p is mandatory when –f is used
-s	--server	<server>	Select netmate server (hostname or IP address)
-t	--stylesheet	<filename>	Select xsl stylesheet file, optional, a default stylesheet is supplied which is used to convert the reply from netmate to ASCII output
-u	--user	<username>	Use that username when connecting to netmate
-V	--version		Show version info and exit
-x	--usessl		Use SSL (secure sockets layer) for control

Short	Long Option	Parameter	Description
			connection; netmate must also run with -x for this to work correctly

Table 3-2: nmrsh command line arguments

3.2.3.3 nmctl

The nmctl script is a wrapper shell script for starting/stopping the netmate tool as a background service. It is intended to be called from within system start/stop scripts but can also be run from the command line. The usage of the script is:

- `nmctl {start|stop|status|restart} [<options and parameters>]`

The commands do the following:

Command	Description
`start`	Start an instance of netmate. Execution of start requires root privileges. This command will output a notification about the success or failure of the starting process. When there already is a netmate running the start command will fail.
`stop`	Stop the currently running netmate. The stop command requires root privileges if netmate was started as user root. A success/fail notification is written to the terminal.
`status check`	Try to connect to a running netmate on localhost and print status information (version, uptime). Without an active netmate the status command will fail.
`restart`	Analogous to nmctl stop ; nmctl start

Table 3-3: nmctl commands

When running start or restart all additional options and parameters (e.g. -i eth0) are passed through to netmate unchanged.

3.2.4 Example Sessions

3.2.4.1 nmctl

There are two ways to run nmctl conveniently from within any directory:

1. Generate a soft link within your file system so that nmctl is available in some directory which is listed within your PATH environment variable:
   ln -s <install_path>/bin/nmctl /usr/sbin/ (you will need to be root to do this).

2. append the installation path to your PATH environment variable by executing:

   `export PATH=${PATH}:<install_path>/bin` (with the Bourne shell and clones) or

   `setenv PATH ${PATH}:<install_path>/bin` (with csh, tcsh and their clones)

After these steps you may need to run "rehash" (tcsh). You can then whether PATH is set correctly by typing:

```
nmrsh  <ENTER>
```

Which should give you this output:

```
Usage: /tmp/netmate_inst/bin/nmctl {start|stop|status|restart} [<netmate_options>]
```

To start an instance of netmate (only one instance allowed per host), type:

```
nmctl start
```

This should give you something like:

```
nmctl: /usr/local/bin/netmate
NetMate version 0.6.1, (c) 2003 - 2004 Fraunhofer Institute FOKUS, Germany
netmate logfile is: /var/log/netmate.log
Up and running.
```

To check the status type:

```
nmctl status
```

which should output something like:

```
nmctl: netmate is running for 99 s, since Tue Dec 16 10:51:43 2003
```

When you get an output saying that the meter is running but do not get the uptime and start time then the script might not be able to connect to netmate and query the status. In that case make sure that the "USER" and "USERPW" variables defined in the nmctl script reflect a user which is allowed to connect to netmate (see section 3.2.5.2 about access control lists).

The command "nmctl stop" terminates a running netmate meter.

3.2.4.2 netmate and nmrsh

Nmrsh is a command line tool for interactive control and querying of netmate by a user (similar to the mysql/mysqld couple). After the initial startup and login to netmate the user is given a command prompt and he/she may interact with the meter. The usual way to start nmrsh is:

```
nmrsh -u <user> [-s <netmate_host>] [--usessl]
```

The hostname can be omitted when netmate is running on the same host as nmrsh is executed. The additional option --usessl needs to be specified if and only if netmate was started with --usessl or the netmate.conf.xml configuration file specifies to use SSL.

The first time a command is entered and committed (pressed RETURN) the user is queried for the access password to netmate before the command is sent. This password is stored and reused by nmrsh for execution of consecutive commands until nmrsh is terminated or netmate has been restarted with a changed access list. Under no circumstances is the password stored externally.

A help text is shown by nmrsh in interactive mode when the user types the command "help" or any other text not recognized by nmrsh as a valid command. It is possible to browse the history of executed commands by pressing the cursor up and down keys. The history will be kept even after nmrsh is terminated and is available on the next invocation of nmrsh again.

For direct execution of commands to netmate the -f or -c options on nmrsh can be used. For details see section 3.2.3.2. In that case there is no interactive shell. The return code of nmrsh will then indicate success of failure of execution of the requested command.

3.2.4.3 netmate and Browser Interface

netmate has a build-in web server giving it a simple web interface (shown in figure below). To use the web interface use a standard browser and connect to the meter using http://host:port/ or https://host:port/ (in case the meter is configured to use SSL) where host is the machine the meter is running on and port is the control port. The web interface can be used to query the meter for information, add and delete rules. Currently the web interface is not very sophisticated and requires some knowledge to use it.

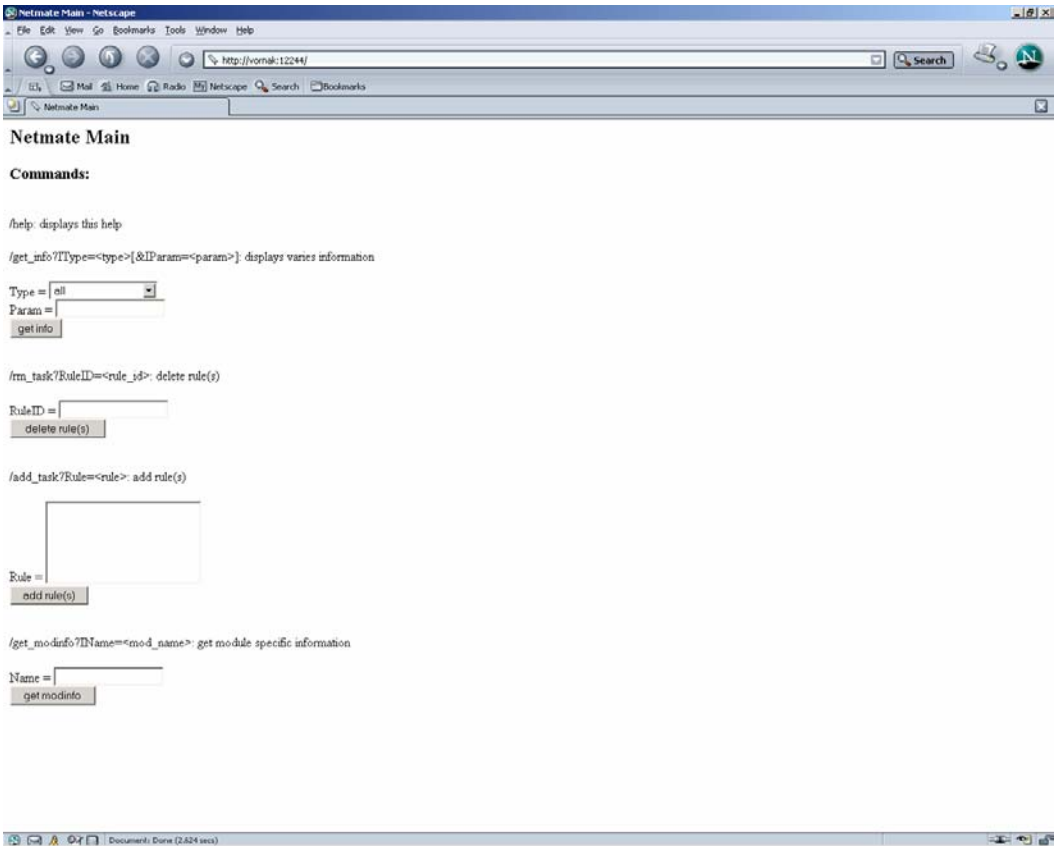

Figure 3-1: netmate web interface

General information can be retrieved by selecting an item from the list and fill in the parameters in the input line. The only information which requires a parameter is “task” which requires the rule name as input. The rule name is <rule set name>.<rule name> as specified in the rule definition.

Rules or rulesets can be deleted. The input must be either a <rule set name> for deleting an entire ruleset or <rule set name>.<rule name> for deleting a single rule.

Rules can be added by typing copying rules into the text area. Rules can be entered in XML format as explained in section 3.2.6.1 or in a format called Meter API (MAPI).

Information about modules requires the input of the module name (as used in rule files).

3.2.5 Customizing NetMate

3.2.5.1 nmrsh Meter Command Syntax

The Meter API is can be used when adding rules to netmate via the nmrsh tool either on the command line (-c parameter) or in interactive mode. The following BFP specification lists the available commands (add_rule, rm_rule, get_info, status) and their parameter syntax:

```
cmd:        add_rule <task_id>
            -r <rulspec>
            -a <action> [<actionopts> [-a <action> <actionopts> ...]]
            -m <mopts>
          | add_tasks <rule_file>
          | rm_rule   <task_id>
          | get_info  [tasklist | status | task=<task_id> | ...]
          | get_modinfo  <module-name>

rulespec:   <filterlist>

filterlist: <filter> [, <filterlist>]

filter:     srcip=<ipaddr/bits> | dstip=<ipaddr/bits>
          | srcport=<int>  | dstport=<int>
          | proto=<ip|tcp|udp>

action:     capture
          | count
          | rtploss
          | pktid    method={plain|crc32} [, base=<bytes>]
          | jitter
          | bandwidth
          | <other modules>

mopts:      start="<datetime>" [, duration=<double/s>
                               [,interval=<double/s>]]

datetime:   YYYY-MM-DD_HH:MM:SS
```

3.2.5.2 Configuration File

The default configuration file read by netmate is the file:

<install_path>/etc/netmate/netmate.conf.xml

A different file can be selected by using netmate's -c option. The default configuration file contains extensive comments. It should be used as a basis for your own configuration file.

The configuration file is written in human readable text format and is structured using XML. It contains five main sections. Each section stores a number of attribute-value pairs and may contain subsections. The five existing sections are:

Section	Description
MAIN	Basic settings related to running netmate
CONTROL	Settings related to the control communication component within netmate such as the access port number and the access control lists
CLASSIFIER	Classification method (algorithm) and parameters can be selected
PKTPROCESSOR	These settings control the behaviour of the packet capture and packet queue (buffer) component. Also parameters for the packet processing modules can be configured here
EXPORTER	These settings control the behavior of the exporting component within netmate. Also parameters for the data export modules can be configured here

Table 3-4: netmate configuration file sections

The sections Classifier, Pktprocessor, and Exporter each have an attribute called "Thread" which can be set to either false or true. This attribute controls whether or not this component of netmate should be run as a separate thread instead of being executed from within the main event loop of netmate. This attribute only has an effect when netmate was compiled with thread support and the system supports Posix threads correctly (pthread library). netmate --version shows whether thread support or SSL support is compiled in.

Note: Whenever an attribute is specified in the configuration file and also on the command line then the command line argument take precedence over the setting in the configuration file.

The following table lists the available settings within the netmate configuration file:

Name	Description	Type	Default
MAIN section			
VerboseLevel	Controls the verbosity of logging to the log file (0 – off, 1 – only severe errors, 4 – standard, 5 – also debug output)	UInt8	4 (0-5 allowed)
LogFile	Name of the log file written by netmate	String	/var/log/ netmate.log
PidFile	Name of process id file used by netmate	String	/var/run/ netmate.pid
FilterDefFile	Name of filter definition file (see 3.2.6.2)	String	filterdef.xml
FilterConstFile	Name of filter constants file (see 3.2.6.3)	String	filterval.xml
NetInterface	Name of network interface to listen on	String	eth0

Name	**Description**	**Type**	**Default**
NoPromiscInt	Can activate non-promiscuous listening on network interface	Bool	no
CONTROL section			
ControlPort	Set port number to listen on	UInt16	12244
UseSSL	Configure whether to accept only control requests encrypted via SSL	Bool	no
UseIPv6	Configure whether to accept control connections via IPv6 (IPv4 connects still possible if set to yes)	Bool	no
LogOnConnect	Set whether control connects are logged	Bool	yes
LogMeterCommand	Set whether commands sent via a control connection are logged	Bool	yes
CLASSIFIER section			
Thread	Choose main loop/extra thread for Classifier	Bool	yes
Algorithm	Select packet classification alrorithm; currently only "Simple" or "RFC"	String	Simple / RFC
SnapSize	Number of bytes to store from captured packet starting at link layer header	Uint16	64
Sampling	Choose sampling algorithm; currently only "All"; more choices within next version	String	All
PKTPROCESSOR section			
Thread	Choose main loop/extra thread for packet processor	Bool	yes
ModuleDir	Directory where netmate looks for packet processing modules (plug-ins)	String	
ModuleDynamicLoad	Choose whether netmate is allowed to load packet processing modules on-demand	Bool	yes
PacketQueueBuffers	Maximum number of packet buffers reserved by netmate's packet queue; the size of the queue (in bytes) is derived automatically	UInt32	20000

Name	**Description**	**Type**	**Default**
Modules	Space-separated list of packet processing modules (e.g. "count bandwidth pktlen") to be loaded by netmate at startup; the list can be empty when 'ModuleDynamicLoad' is true	String	
EXPORTER section			
Thread	Choose main loop/extra thread for data exporter	Bool	yes
ModuleDir	Directory where netmate looks for data export modules (plug-ins)	String	
ModuleDynamicLoad	Choose whether netmate is allowed to load data export modules on-demand	Bool	yes
Modules	Space-separated list of data export modules (e.g. "text_file bin_file") to be loaded by netmate at startup; the list can be empty when 'ModuleDynamicLoad' is true	String	

Table 3-5: configurable parameters in netmate configuration file

3.2.5.2.1 Netmate Control Access List

Part of the CONTROL section is the ACCESS subsection. This subsection configures which users are allowed to connect to and control/query netmate from which hosts. There are two types of entries: ALLOW and DENY. The type of an entry can be either "Host" or "User". In case of "Host" the value of that entry needs to be an IPv4 or IPv6 address or a valid hostname (including domain name). In case of "User" the value of that entry must be a user name and optionally a password. The two items have to be written as "USERNAME:PASSWORD". The special value "All" may be used at the end of the list used to block all non-listed users and hosts.

3.2.5.2.2 Packet Processing Modules Section

In the PKTPROCESSOR section there is a subsection called MODULES. In that subsection the user may specify default parameters which shall be passed to specific modules. The possible attributes are specific to each module and are explained in section 3.2.8

3.2.5.2.3 Data Export Modules Section

In the EXPORTER section there is a subsection called MODULES. In that subsection the user may specify default parameters which shall be passed to specific modules. The possible attributes are specific to each module and are explained in 3.2.9

3.2.5.3 Control Protocol Response

Control protocol replies returned from netmate as answer to a request are based on an XML template which is sent over HTTP. The template is simple and contains two parts: status and message part. The @status@ and @message@ strings are replaced by netmate before the reply is sent (similar to autoconf substitution). The XML around the keywords can be changed. After changing the reply template make sure that the XSLT translations to HTML and text still work (see chapter below).

The template file is etc/reply.xml.

3.2.5.4 Web GUI Template

As explained above the meter can be controlled with a standard web browser. The meter can be accessed by simply connection to the following URLs: `http://host:port/` or `http://host:port/help` (where host is the host where netmate is running and port is the control protocol port configured). In this case netmate will respond with its main page which is a plain HMTL file and can be changed.

The main page is <prefix>/etc/netmate/main.html

3.2.5.5 XSLT Files

To translate the XML replies into HTML or readable text netmate uses xslt transformation files. One file is needed to translate to HMTL (browser based control) and one file is needed by nmrsh (shell based control). If the reply XML template is changed or the control protocol is extended these files must be adapted.

The XSLT files are <prefix>/etc/netmate/reply.xsl (HTML output) and <prefix>/etc/netmate/reply2.xsl (plain ASCII output).

3.2.6 Rule Definition

To work with netmate the user must define measurement rules. A number of rules is called a rule and is defined in a ruleset file. Measurement rules define what to measure. A rule consists of four parts: filter, action, export and miscellaneous part. The following is an example rule:

```
<RULE ID="example_rule">
    <FILTER NAME="SrcIP">*</FILTER>
    <FILTER NAME="DstIP">www.google.com</FILTER>
    <FILTER NAME="DstPort">http</FILTER>

    <ACTION NAME="bandwidth">
    </ACTION>

    <EXPORT NAME="text_file">
        <PREF NAME="Filename">example_rule_data</PREF>
    </EXPORT>

    <!-- match both directions of the flow -->
    <PREF NAME="bidir">yes</PREF>
    <!-- start 10 seconds later -->
    <PREF NAME="Start">+10</PREF>
    <!-- overwrite default settings -->
    <PREF NAME="Duration">990</PREF>
    <PREF NAME="Interval">5</PREF>
</RULE>
```

The rule has the name or identifier “example_rule”. The filter part defines that all packets match the rule that are send from any source to the HTTP port on “www.google.com”. The

action part defines that the meter will count all packets and will compute the average bandwidth. The export part specifies that the flow data will be exported using the module "text_file". This means the data are exported into a text file with the name "example_rule_data". The miscellaneous part of the rule defines that this is a bidirectional rule matching packets in both directions. This means not only the packets going to the HTTP port of "www.google.de" are matched but also the packets coming from that port on the host. The further settings define that the rule will be activated 10 seconds after it is loaded into the meter, it then will be active for 990 seconds and it will export metric data (counters, bandwidth) every 5 seconds.

The rule shown above is written in exactly the way it would appear as part of a ruleset file. The action, export and miscellaneous part are straightforward. Action and export list a number of action or export modules. The name must equal the filename of the shared library (except the extension .so). Each module may support multiple configuration options (preferences) which is described in the module's documentation. The "tex_file" module has a configuration option called "Filename". The miscellaneous part contains the preferences of the rule. The preferences most used are: Start, Stop, Duration, Interval and bidir. Their meaning will be explained in the next section. The filter part is the most complicated. To allow a user maximum flexibility in defining rule filters netmate does not use a fixed set of filters as other tools do (e.g. tcpdump). Instead it allows the user to define his filters in a filter attribute definition file. A filter definition will define a filter as tuple of referrer, offset, length, type, name and optionally mask. For all types of fixed length the length attribute can be omitted because it is implicitly set by the meter itself during the parsing. Filter definitions are explained in detail in section 3.2.6.2. After defining a filter attribute it can then be used in ruleset definitions. Because it is more convenient to use port names etc. instead of port number netmate also supports a flexible way of defining filter values. Filter values are specified in a filter value file as tuple of type, name and value. All defined values can be referenced by their name in a ruleset and the meter will automatically replace the name with the value during the rule parsing. The figure below shows the interworking of the three different definition files.

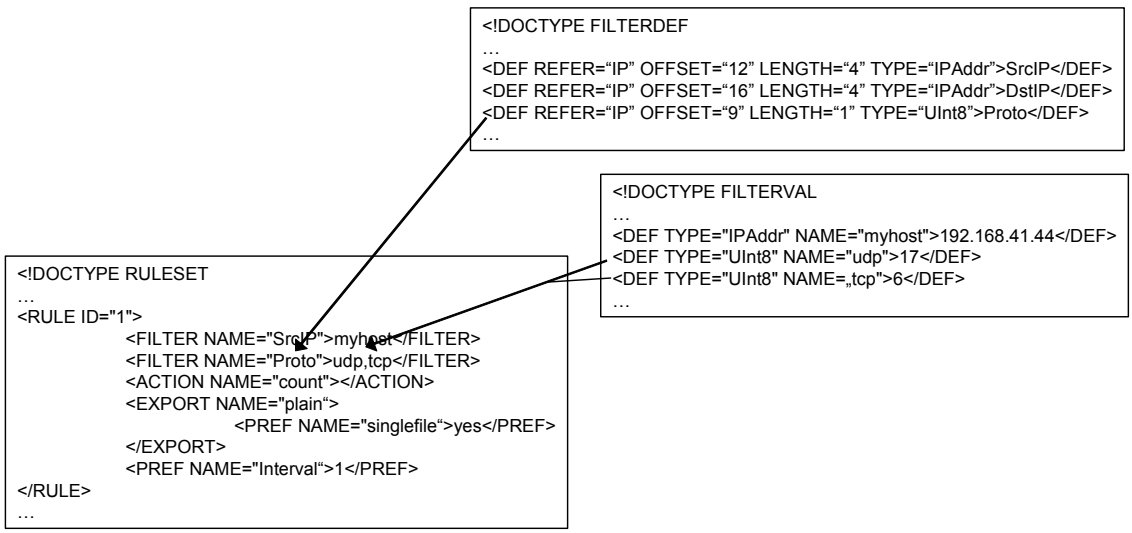

Figure 3-2: Rule Definition

In the following we explain how to write rulesets, filter attribute definitions and filter value files.

3.2.6.1 Rule File

The following shows an example ruleset file interspersed comments.

The first two lines are always the following:

```
<?xml version ="1.0" encoding="UTF-8"?>
<!DOCTYPE RULESET SYSTEM "rulefile.dtd">
```

Then a line follows defining an identifier for the set of rules. The set ID can be used for example to delete a whole set of rules (instead of deleting all rules separately). The identifier can be an arbitrary string containing number, characters (except the “).

```
<RULESET ID="set_id">
```

The first part of a ruleset is the global part. All settings defined in the global part will be used for all rules in the set. It is possible however to overwrite a global setting with a rule specific setting. Preferences can be overwritten by having the same preference in a rule definition with a different value. Similar action and export modules are overwritten if they appear again in a rule. Global preferences for a module will not become preferences for a module specific as part of a rule! All preferences for a module specified in a rule must be explicitly defined e.g. if a global setting is to use the “tex_file” export module with a Filename parameter, a later use of “text_file” export within a rule must have the same Filename preference if the metric data shall be exported in the same file.

In this example the global part specifies that each rule will use the “show_ascii” processing module and all rules will be exported to text files.

```
<GLOBAL>
  <!-- defaults for all rules can be overwritten by rule
       specific configuration -->

  <!-- how long the rules are active (in seconds) -->
  <PREF NAME="Duration">1000</PREF>
  <!-- export interval -->
  <PREF NAME="Interval">10</PREF>

  <ACTION NAME="show_ascii"></ACTION>

  <EXPORT NAME="text_file">
    <!-- by default each rule is exported into a separate file
         named like the rule ID; the following line can be uncommented
         to export all rules into a single file-->
    <!--<PREF NAME="Filename">exportfile</PREF>-->

    <!-- the following can be uncommented to create one file per action
         per rule -->
    <!--<PREF NAME="Multifile">yes</PREF>-->

    <!-- specify user -->
    <!--<PREF NAME="ExportUser">nobody</PREF>-->
  </EXPORT>
</GLOBAL>
```

After the global part the rule definitions start. Each rule in a set must have a unique ID.

```
<RULE ID="1">
  <!-- single masked value -->
  <FILTER NAME="SrcIP" MASK="255.255.255.0">193.175.133.0</FILTER>
  <!-- list of values -->
  <!-- FILTER NAME="Proto">udp,tcp</FILTER -->

  <ACTION NAME="bandwidth">
    <PREF NAME="Interval">5</PREF>
  </ACTION>

  <ACTION NAME="pktlen">
  </ACTION>
```

```
  </RULE>

  <RULE ID="2">
    <FILTER NAME="SrcIP">*</FILTER>
    <FILTER NAME="Proto">udp,tcp</FILTER>
    <!-- range -->
    <FILTER NAME="DstPort">1-1024</FILTER>

    <ACTION NAME="jitter">
    </ACTION>

    <ACTION NAME="port_use">
    </ACTION>
  </RULE>

</RULESET>
```

The XML document definition is shown in the design document. After discussing the example of a ruleset we know explain in detail how to define rules.

3.2.6.1.1 Preferences

Preferences are configuration settings. Preferences can be either global, rule-specific or action/export module specific. The syntax for preferences is:

```
<PREF NAME="preference-name" TYPE="pref-type">pref-value</PREF>
```

The preference-name identifies the preference. Preferences names are case sensitive. The table shows the existing preferences and their meaning. All can be used in the global section or in the rule specific part.

Name	**Description**	**Value Format**	**Global**	**Rule-specific**
Start	Activation time of the rule	YYYY-MM-DD HH:MM:SS | + offset in seconds Default is loading time	yes	yes
Stop	Deactivation time of the rule	YYYY-MM-DD HH:MM:SS Default is no stop time	Yes	yes
Duration	Duration of the rule	Duration in seconds Default is 0	Yes	yes
Interval	Export interval [s]	Interval value in seconds (default is 0 meaning export upon rule deactivation only)	yes	yes
bidir	Rule is bidirectional (can be enforced by a processing module!)	yes | no (default)	yes	yes
Align	Align export on	yes | no (default)	yes	yes

Name	Description	Value Format	Global	Rule-specific
	interval time slots			
FlowTimeout	Flow timout enabled	yes | no (default) | timeout value	yes	yes

Table 3-6: task attributes

A rule must have either a stop time or duration. If metric data shall be exported during runtime the rule must have the export interval set.

The following types can be used for preferences: UInt8, SInt8, UInt16, SInt16, UInt32, SInt32, UInt64, SInt64, Bool, Binary, String, IPAddr, IP6Addr. If no type has been specified the value is assumed to be of type String. The type information is used by netmate to do type checking during parsing.

Module specific preferences depend on the module. Section 3.2.8 and 3.2.9 explain the module specific preferences.

3.2.6.1.2 Filter

Filters are used in a rule to define what packets a rule matches. The syntax for filters is:

```
<FILTER NAME="filter-name" MASK="filter-mask">filter-value</FILTER>
```

Filter-name must be one of the filters specified in the filter attribute definition file. Filter names are case insensitive! Filter-mask is an optional mask which is applied to the packet field before matching. The filter-value can be an exact match, range match, set match or wildcard. The support of masks and filter types depend on the classification algorithm used (see classifier section). The following tables summaries the different filter types.

Filter Type	Value Format	Example
Exact	value	192.168.41.1
Range	low_value - high_value	1 - 1024
Set	value1, value2, … , valueN (maximum of 16)	tcp, udp, icmp
Wildcard	*	*

Table 3-7: filter types

A rule can have an arbitrary number of filters. If more than one filter is specified for a rule the filters are logically AND'ed. Range and Set matches are a way of logically OR'ing filtes.

Examples:

```
<FILTER NAME="DstPort">*</FILTER> <!-- macthes all destination ports -->
```

```
<FILTER NAME="Proto">udp,tcp</FILTER> <!-- matches udp or tcp packets -->
<FILTER NAME="SrcIP">192.168.41.1</FILTER> <!-- matches all packets from
192.168.41.1 -->
<FILTER NAME="SrcPort">1-1024</FILTER> <!-- matches all packets from port
1-1024 -->
<FILTER NAME="SrcIP" MASK="255.255.0.0>192.168.0.0</FILTER> <!-- matches
all packets from 192.168 network -->
```

3.2.6.1.3 Actions

Actions specify the metrics to calculate. Actions are realized as modules which are dynamically loaded by the meter (shared libraries). The syntax for actions is:

```
<ACTION NAME="bandwidth">

<!– action parameters -->

</ACTION>
```

The action name must be the module name with the extension. Module names are case sensitive. Preferences can be set for an action (see processing modules section)

3.2.6.1.4 Export

Exports specify how the metric data (flow records) are exported by the meter. Exports are realized as modules which are dynamically loaded by the meter (shared libraries). The syntax for exports is:

```
<EXPORT NAME="text_file">

<!–export parameters →

</EXPORT>
```

The export name must be the module name with the extension. Module names are case sensitive. Preferences can be set for an export (see export modules section)

3.2.6.2 FilterDef File

The filter attribute definition file defines the filter attributes which can be used in rules. The following is an example (part) of a filter definition file. The file start with a standard header

```
<?xml version ="1.0" encoding="UTF-8"?>
<!DOCTYPE FILTERDEF SYSTEM "filterdef.dtd">
<FILTERDEF>
```

The main part of the file is a list of filter definitions:

```
  <!-- Ethernet -->
  <DEF REFER="MAC" OFFSET="0" LENGTH="6" TYPE="Binary">DstMAC</DEF>
  <DEF REFER="MAC" OFFSET="6" LENGTH="6" TYPE="Binary">SrcMAC</DEF>

  <!-- IP -->
  <DEF REFER="IP" OFFSET="0" MASK="0xF0" TYPE="UInt8">IPVer</DEF>
  <DEF REFER="IP" OFFSET="1" TYPE="UInt8">IPToS</DEF>
  <DEF REFER="IP" OFFSET="8" TYPE="UInt8">IPTTL</DEF>
```

```
  <DEF REFER="IP" OFFSET="9" TYPE="UInt8">Proto</DEF>
  <DEF REFER="IP" OFFSET="12" TYPE="IPAddr" REV="DstIP">SrcIP</DEF>
  <DEF REFER="IP" OFFSET="16" TYPE="IPAddr" REV="SrcIP">DstIP</DEF>

  <!-- ICMP -->
  <DEF REFER="TRANS" OFFSET="0" TYPE="UInt8">ICMPType</DEF>
  <DEF REFER="TRANS" OFFSET="1" TYPE="UInt8">ICMPCode</DEF>

  <!-- UDP, TCP -->
  <DEF REFER="TRANS" OFFSET="0" TYPE="UInt16" REV="DstPort">SrcPort</DEF>
  <DEF REFER="TRANS" OFFSET="2" TYPE="UInt16" REV="SrcPort">DstPort</DEF>
  <DEF REFER="TRANS" OFFSET="13" MASK="0x3F" TYPE="UInt8">TCPFlags</DEF>

</FILTERDEF>
```

The syntax of a filter definition is:

```
<DEF REFER="reference" OFFSET="offset" TYPE="type" LENGTH="length"
MASK="mask" REV="reverse-attribute">filter-attribute-name</DEF>
```

The reference is the initial offset to the layer in which the attribute is located. Reference can be: MAC (Layer2), IP (Layer3), TRANS (Layer4) or DATA (Layer5-7). The refernce is necessary because a number layer headers (e.g. IP, TCP) are of variable size and a fixed offset would not work Offset is the offset of the attribute in bytes from the reference point. Type specifies the type of the attribute. The following types can be used for preferences: UInt8, SInt8, UInt16, SInt16, UInt32, SInt32, UInt64, SInt64, Binary, String, IPAddr, IP6Addr. Length is the length of the attribute in bytes. Length can be omitted for all fixed size types but must be given for variable size types such as Binary and String.

Example: if reference is IP, offset is 0 and type is UInt8 than the length is 1 and the attribute is the first byte of the IP header.

The optional mask is applied to the packet value before matching (this means that if the filter attribute definition and the filter in a rule both have masks specified both masks are applied to the packet value before matching).

The optional reverse attribute is must be defined for bidirectional matching and specifies the reverse attribute (i.e. the attribute is exchanged by the reverse attribute in the opposite direction).

The XML document definition is shown in the design document.

Examples:

```
<DEF REFER="TRANS" OFFSET="6" TYPE="UInt16">UDPChecksum</DEF>
```

3.2.6.3 Filter Value File

The filter value file specifies values (constants) which can be used as filter values in rule files. The following is an example (part) of a filter value file. The file start with the standard XML header:

```
<?xml version ="1.0" encoding="UTF-8"?>
<!DOCTYPE FILTERVAL SYSTEM "filterval.dtd">
```

```
<FILTERVAL>
```

The main part contains a list of filter values:

```
  <!-- my constants -->
  <DEF NAME="myhost" TYPE="IPAddr">193.175.133.179</DEF>

  <!-- constants from /etc/protocols -->
  <DEF NAME="ip" TYPE="UInt8">0</DEF>
  <DEF NAME="hopopt" TYPE="UInt8">0</DEF>
  <DEF NAME="icmp" TYPE="UInt8">1</DEF>
  <DEF NAME="igmp" TYPE="UInt8">2</DEF>
  <DEF NAME="ggp" TYPE="UInt8">3</DEF>
  <DEF NAME="st" TYPE="UInt8">5</DEF>
  <DEF NAME="tcp" TYPE="UInt8">6</DEF>

</FILTERVAL>
```

The syntax for a filter value is:

```
<DEF NAME="filter-value-name" TYPE="type">filter-value</DEF>
```

The filter value name defines the name which can be referenced from the rule. The type defines the type of the value and can be one of the following: UInt8, SInt8, UInt16, SInt16, UInt32, SInt32, UInt64, SInt64, Binary, String, IPAddr, IP6Addr. Finally the filter value is the value by which the name is replaced in the rule. A filter value must be of the same type as the filter attribute!

The XML document definition is shown in the design document.

Examples:

```
<FILTER NAME="Proto">udp,tcp</FILTER> <!-- match UDP only -->

<FILTER NAME="Proto">udp,tcp</FILTER> <!-- match UDP or TCP -->
<FILTER NAME="SrcPort">udp</FILTER>   <!-- WRONG: SrcPort is UInt16 and udp
is UInt8 -->
```

3.2.7 Packet Classifiers and Samplers

The packet classification is the process of assigning all matching rules to an incoming packet. This is done by applying the filters of all rules which are defined in the ruleset. The classifier does support any dynamic module loading mechanism because we expect that most users are satisfied with the provided classifier. Currently netmate support two different classifiers.

The simple classifier is a simple linear list of rules. It basically has no restrictions concerning the number of rules. It uses few memory (O(N) complexity) but the classification can be slow if the number of rules becomes large (O(N) complexity).

The RFC classifier is based on the idea of Recursive Flow Classification (GuMc99). It supports only up to 1000 rules because of the huge memory requirements ($O(N^k)$ complexity). The classification performance is fast ($O(\log_{k-1}N)$ complexity). In the current implementation it is independent of the number of rules.

(N = number of rules, k=number of dimensions (filters))

The classifier can be selected via the configuration file using the Algorithm preference which can be either “Simple” or “RFC”.

Sampling is the process of selecting only specific packets before classification. netmate does only support sampling before classification. It does not support sampling after classification (flow-based sampling). Currently it only supports exactly 1 sampling function. At the moment only one sampling class has been implemented which samples all incoming packets.

The sampler can be selected via the configuration file using the Sampling preference which can be currently only “All”.

3.2.8 Packet Processing Modules

Netmate can analyse incoming packets in a variety of ways starting with simple packet and volume accounting up to detailed application level analysis (e.g. for http requests packets).

Such a packet processing module is implemented as a binary code plug-in for netmate and can be loaded dynamically at runtime. By default modules are loaded on-demand at runtime when they are first referenced by some meter task. When netmate cannot load a module it returns with an error message and the task is rejected.

Each task may use a different module. Also all tasks may use the same module. For example if all tasks make use of the bandwidth module then netmate maintains a separate set of counters for each task. In addition a single task may use any number/combination of packet processing modules, e.g. “bandwidth” and “pktlen” and “show_ascii”.

For each of the supported traffic metrics a different packet processing module is responsible. In the current distribution the netmate toolset comes with the following packet processing modules:

Module	**Description**
bandwidth	Count packets, traffic volume and compute packet rate plus bandwidth per analysis interval. By default the analysis interval is equal to the data export interval. It can be set to a shorter value via the module parameters.
capture	This module is intended to be used in conjunction with the binary export module and enables netmate to produce a tcpdump trace files per task.
count	Most basic module, only counts packets and data volume (cumulative; counters are not reset on data export)
jitter	Compute min/max/average and variance of packet inter arrival times
pktid_crc32	Compute unique packet IDs based on packet header and part of packet content using a CRC32 algorithm
pktid_md5	Compute unique packet IDs based on packet header and part of packet content using an MD5 algorithm
pktlen	Logs min/max/average length of observed packets per export interval
port_use	Account and log the most frequently used application protocols (i.e. UDP and TCP destination port numbers) with number of packets and bytes within the flow observed by a task

Module	Description
rtploss	Observe rtp (real-time protocol) packets and compute the observed packet loss based on the consecutive packets numbers in the rtp headers
rtt_ping	Compute round packet round trip times based on ICMP ping packets and answers. This requires the filter to specify proto=icmp and it automatically enforces the bidir option
show_ascii	This module analyses IP and UDP/TCP packet headers and exports many of the header fields such as src/dst address, ttl, src/dst port number and others
testmod	This module is intended to be the basis for newly developed packet processing modules. For details about building your own packet processing module read section 4.1 carefully

Table 3-8: available packet processing modules

3.2.9 Data Export Modules

Netmate can export information produced by traffic analysis (i.e. packet processing modules) in a variety of ways. Netmate features the concept of loadable modules for data export as well as for packet processing.

Note that modules for traffic analysis and modules for data export are completely independent from one another and can be combined as desired. This means that for instance the "text_file" data export module can be applied to all kinds of traffic analysis data. This concept is explained in detail in the development section 4.2.

When specifying metering tasks the user needs to select a module to use for data export explicitly. A default module can be specified to be used by all tasks. In addition each task may use a different data export module (e.g. one task uses "text_file" and another uses "bin_file"). Any task can also export via a set of modules (e.g. "text_file" and "bin_file").

The following data export modules are currently shipped with netmate:

Module	Description
text_file	writes information from traffic analysis into human readable ASCII files on hard disk (or any other locally mounted file system). An example output is shown in the appendix at section 6.4.5.1)
bin_file	writes information from traffic analysis as binary data (in network byte order) to a files on hard disk (or any other locally mounted file system). Currently this module is mostly used in conjunction with the "capture" packet processing module and can produce tcpdump [tcpdump] compatible packet trace files.
testexp	this module is intended as a basis for development of future export modules (e.g. direct transport of analysis data via some protocol to a remote host). More detailed information about is can be found in section 4.2)

4 Extending NetMate

4.1 New Packet Processing Modules

In order to extend netmate with new traffic analysis functions developers can write their own packet processing modules. Such modules are implemented in C. They all implement the functions of the Packet Processing Module API (PPM-API) and thus can be loaded by netmate at runtime as binary plug-ins.

New modules can be easily compiled and installed when you have checked out the development version of the netmate toolset via CVS or downloaded a source package. In order to build a new packet processing module first go to the directory

```
netmate/src/proc_modules
```

and copy the file "testmod.c" to "newmod.c". Then replace all occurrences of "testmod" within "newmod.c" with "newmod". You will need to modify the Makefile.am within the proc_modules directory and rerun ./configure from the top directory "netmate" to have your new module compiled automatically when executing "make".

New processing modules must be included in the Makefile.am under lib_LTLIBRARIES. Furthermore the following lines must be added to the file:

```
<module-name>_la_LDFLAGS = -export-dynamic -module
<module-name>_la_SOURCES = <module-name>.c
<module-name>_la_LIBADD = ProcModule.lo
```

Now you are ready to open "newmod.c" in your favourite editor and implement the methods of the PPM-API to realise your traffic analysis functions. It is also a good idea to open a simple module such as pktlen.c for reference when developing a new module.

The lifecycle of a module is as follows:

	Event	**methods called**
1	Module gets loaded by Module Loader	initModule, getTypeInfo, getModuleInfo
2	New rule(s) gets added	initFlowRec, parseConfig
3	Rule(s) deleted	destroyFlowRec
4	Flow timeout	timeout
5	Module specific timeout	timeout
6	Export data for rule	exportData
7	Packet arrival	processPacket
8	Module gets unloaded	destroyModule

The methods of the PPM-API and their meaning are as follows:

Name	Description
initModule	initialise the module; called only once directly after loading the module into netmate. It is used to init variables and memory which are shared by all tasks using this module
destroyModule	cleanup before the module gets unloaded; called only once directly before unloading the module. A module is unloaded when netmate terminates or when there is no more task referencing that module. Its task is to cleanup data structures (free memory) reserved within initModule
initFlowRec	initialise a particular flow record; called whenever a new task makes use of this module. This method must reserve a portion of memory needed to store intermediate data (such as volume counters) for exactly this task and module. The memory is required to be set to some initial values and is then handed over to netmate which manages all flow records.
destroyFlowRec	cleanup flow record; opposite of initFlowRec method; called whenever a task referring that module terminates. The method is required to free the flow record and any allocated subcomponents relating to that task and module. Afterwards the task is removed from netmate.
resetFlowRec	reset the flow record; this method may be called by netmate after data collection when a task requested that the module shall start working from the start after each data export (e.g. for relative packet counters)
parseConfig	parse configuration parameters given for this module; called when a new task is initiated after the call to initFlowRec. This method receives all attribute-value pairs specified for this task and module from the configuration (either rule set file or add_task command). The method must parse the parameters and shall store values in the flow record for later use (if needed).
processPacket	process a packet; called for each incoming packet matching the filter description of a particular task. This method receives the raw packet data, metadata (timestamp, packet length, etc.) and the current flow record for the task and module. It must modify the flow record accordingly based on values from metadata and packet data.
exportData	export measurement data for a flow; called whenever an export interval for a particular task has expired. This method is required to write the information from the current flow record in a netmate-internal format (see for instance pktlen.c::exportData) into a local memory buffer and hand over access to that buffer to netmate. The exported data must adhere to the specification of exported data as it is provided by the module by means of the exportInfo data structure. This data structure is explained in detail in section 4.1.1 .
timeout	callback function if timer was set; a module may specify an arbitrary number of (one-time or recurring interval) timers for notification or

Name	Description
	watchdog purposes. These timers are independent from the data export interval timer. Timers are initialised per rule using that module and may be set depending on the task parameters in the parseConfig function (see bandwith.c for an example use of timers).
getTypeInfo	provides the type info for the data generated; method provided by the netmate system; do not overload with own implementation
getModuleInfo	provide information about the module; this method shall return information about the functions and the author of the module (see bandwith.c for a complete example).
getErrorMsg	return last error in textual format; whenever one of the other functions returns with an error code (return value not equal zero) then this method is called with the error code to get a human readable error description.

Table 4-9: packet module API methods

4.1.1 Runtime type information

Each packet processing module must specify what information it exports. For this reason the programmer of a packet processing module must include in the source code a structure which notes, in the form of a table, what the values and the data types of the exported values are. The first value of an entry in the type specification denotes the data type and the second value gives an (arbitrary) name for the exported value.

For the `count` module the data structure looks like this:

```
typeInfo_t exportInfo[] = {
      { UINT32, "packets"    },
      { UINT32, "volume"     },
      { UINT32, "first_time" },
      { UINT32, "first_time_us" },
      { UINT32, "last_time"  },
      { UINT32, "last_time_us"  },
      EXPORT_END };
```

This list specifies that the count module exports six unsigned 32 bin integers per export record and that the meaning of these six values correspond to number of packets, number of bytes and the timestamps of the first and last observed packet. In this way a module may only export one datasets per export interval for a rule. The count module only ever exports one dataset of six integers per export interval.

Numerous data types are possible as the following example shows:

```
typeInfo_t exportInfo[] = {
    { CHAR, "a_char"  },
    { INT8, "int8_1"  },
    { INT8, "int8_2"  },
    { INT16, "name16" },
    { INT32, "name32" },
    { INT64, "name64" }
EXPORT_END };
```

The defined and supported data types are:

```
CHAR, INT8, INT16, INT32, INT64, UINT8, UINT16, UINT32, UINT64, STRING,
BINARY, IPV4ADDR, IPV6ADDR, FLOAT, DOUBLE
```

These are defined in the source file `src/include/ProcModuleInterface.h`

A more complicated module for instance a round trip calculation module may need to export a timestamp and a delay value and then export a variable length list per export interval where each row corresponds to one observed RTT measurement during that measurement interval. The export of a list of data records is supported via the LIST export type:

```
typeInfo_t exportInfo[] = {
    { LIST, "list" },
    { UINT16, "col1"  },
    { UINT16, "col2"  },
    { UINT32, "col3"  },
    { UINT8, "col4"  },
    LIST_END,
    { LIST, "list2" },
    { UINT16, "an_id" },
    { STRING, "a_text" },
    LIST_END,
EXPORT_END };
```

In this example the module specifies to export two lists whenever an export is due fir a task. In is expected to write n times (two 16 bit, one 32 bit, 1 8 bit) values plus m times (one 16 bit value and a string). Upon export n and/or m may be zero. For an example as to how to write those values at export have a look at the file `src/procmodules/testmod.c`, method `exportData`.

4.1.2 Module Timers

Packet processing modules are usually only called for a rule whenever a rule receives a matching packet. In addition a module may specify timers in order to be called at regular intervals even when no packets have been received. These timers allow for instance the convenience programming of a watchdog functionality, i.e. a module that exports an alert notification when *no* packets of a specific flow have been observed for a certain time.

Each module can specify a number of timers via a list structure in the following format (example):

```
timers_t timers[] = { /* handle, ival_msec, flags */
    { 1, 1000, 0 },
    { 2, 2000, TM_RECURRING },
    { 3, 3000, TM_ALIGNED },
    { 4, 4000, TM_RECURRING | TM_ALIGNED },
    { 5, 5000, TM_NONE /*==0*/ },
    TIMER_END
};
```

This specifies that the module wants five timers each with a different timer id, interval, and behaviour. Timers without the TM_RECURRING flag set are only executed once and then disabled. Timers with the TM_ALIGNED flag set have their times aligned to their interval.

This means that for instance a timer with interval = 60000 (60 seconds) will be called every full minute at mm:00 even if the task was started at 12:10:30.
Please note that the set of timers is initiated *per task* using that module! All such timers are managed by netmate in a central time calendar. Note however that using many tasks with modules that user timers may still cause a performance problems in case the timer intervals are very short and the timers are of recurring type.

4.2 New Export Modules

In order to extend netmate with new data export methods, developers can write their own data export modules. Such modules are implemented in C++. They all implement the functions of the Data Export Module API (DEM-API) and thus can be loaded by netmate at runtime as binary plug-ins.

New modules can be easily compiled and installed when you have checked out the development version of the netmate toolset via CVS or downloaded a source package. In order to build a new packet processing module first go to the directory

```
netmate/src/export_modules
```

and copy the file "testexp.c" to "newexp.c". Then replace all occurrences of "testexp" within "newexp.cc" with "newexp". You will need to modify the Makefile.am within the export_modules directory and rerun ./configure from the top directory "netmate" to have your new module compiled automatically when executing "make".

New export modules must be included in the Makefile.am under lib_LTLIBRARIES. Furthermore the following lines must be added to the file:

```
<module-name>_la_LDFLAGS = -export-dynamic –module
<module-name>_la_SOURCES = <module-name>.c
<module-name>_la_LIBADD = ExportModule.lo
```

Now you are ready to open "newexp.cc" in your favourite editor and implement the methods of the DEM-API to realise your new data export method. It is also a good idea to open a the standard module text_file.cc for reference when developing a new export module.

The lifecycle of a module is as follows:

	Event	**methods called**
1	Module gets loaded by Module Loader	initModule, getInfo
2	New rules(s) get added	initExportRec
3	Rule(s) deleted	destroyExportRec
4	Export protocol packet received	handleFDEvent
5	Module specific timeout	timeout
6	Export due	exportData
7	Module gets unloaded	destroyModule

Export modules are incorporated by the Exporter component of netmate. This list shows the interface of an export module:

Method	Description
initModule	initialise the module
destroyModule	cleanup before the module gets unloaded
initExportRec	init a particular flow record
destroyExportRec	cleanup flow record
exportData	export measurement data for a flow
timeout	callback function if timer was set
handleFDEvent	handle event on file descriptor this module own
getInfo	provide information about the module
getError	return last error

4.3 New Packet Classifier

New classifiers can be written by implementing a new class which is derived from the Classifier class. A classifier must implement at least the following methods:

```
//! constructor

MyClassifier( ConfigManager *cnf, string name, NetTap *nt = NULL, Sampler
*sa = NULL, PacketQueue *queue = NULL, int threaded = 0 );

//! destructor

virtual ~MyClassifier();

//! check a ruleset (the filter part)

virtual void checkRules(ruleDB_t *rules) = 0;

//! add rules

virtual void addRules(ruleDB_t *rules) = 0;

//! delete rules

virtual void delRules(ruleDB_t *rules) = 0;

//! classify an incoming packet (determine matching rule(s))
```

```
virtual int classify(metaData_t* pkt) = 0;

//! Called if new incoming packet is detected

virtual int handleFDEvent(eventVec_t *e, fd_set *rset, fd_set *wset,
fd_sets_t *fds);

//! called initially as the thread inside the Classifier (if the classifier
runs in a separate thread)

virtual void main();
```

Apart from implementing a new classifier the constructor in Meter.cc must be extended to allow the use of the new classifier via the configuration file.

Note: New classifier source and header files must be includes in the Makefile.am under netmate_SOURCES.

4.4 New Sampling Algorithms

A new sampling method can be implemented by deriving a new sampling class from the base Sampler class. A sampler class must implement the following functions:

```
//! constructor

MySampler();

//! destructor

virtual ~Sampler();

//! \returns 1 if packet sampled 0 otherwise

virtual int sample(metaData_t* pkt) = 0;
```

The sampling function has access to the whole packet including the meta data such as timestamp etc.

A new sampling class also requires changes in the constructor of Meter.cc to be able to select the new sampler via the configuration file.

Note: New sampler source and header files must be includes in the Makefile.am under netmate_SOURCES.

4.5 Overall Architecture

The software is based on C++ and has a very modular design (hopefully). Users who want to change part(s) of the core implementation should read the design document first.

4.6 Debugging

For debugging the netmate toolkit it is required to compile the tools with debugging support. This enables a much more verbose output to the logging file. It also adds symbol information to the binary needed for running it within a debugger.

To add debugging support fetch the source package from the web and configure the compilation process (see section 3.1.1) with additional debug support by adding:

```
--enable-debug
```

to the call to the ./configure script. Then compile (make) and install the meter software. The debug version can be used/controlled in exactly the same way as the non-debug version.

We use valgrind [valgrind] for memory allocation debugging. Please read the valgrind documentation. Note that the C++ library has its own memory pools valgrind will report memory still reachable after exiting netmate. C++ memory optimization can be disabled with setting the GLIBCPP_FORCE_NEW environment variable. For more information read the valgrind FAQ.

4.7 Future Plans

It is planned to extend the netmate toolset in the future with additional features such as:

- SQL export module
- IPFIX export module
- more packet processing modules (e.g. flow detection, RTT computation)
- integration of packet sampling methods

4.7.1 Feedback

If you have any problems, bug reports, patches or other feedback please contact:

- Sebastian Zander (zander@fokus.fraunhofer.de)
- Carsten Schmoll (schmoll@fokus.fraunhofer.de)

5 References

gcc/g++	GNU C/C++ compiler	Refer to your operating system documentation how to obtain and install gcc
valgrind	System for debugging and profiling x86-Linux programs	http://valgrind.kde.org/
cachegrind	valgrind add-on for memory and performance profiling	http://www.cs.swarthmore.edu/local/debugging/valgrind/cg_main.html
doxygen	Documentation system for C/C++	http://www.stack.nl/~dimitri/doxygen/
readline	Library for building user interactive convenient shell-like programs	http://cnswww.cns.cwru.edu/~chet/readline/rltop.html
libpcap	Berkeley packet capture library	http://www.tcpdump.org
libxml2, libxslt	Standard libraries for processing XML documents and performing XSLT transformations	http://xmlsoft.org
libcurl	Library for HTTP transfers (clients)	http://curl.haxx.se
openssl	Secure sockets layer library	http://www.openssl.org
pthreads	POSIX threads library	Refer to your operating system documentation how to obtain and install the pthreads library
rtfm	Real-time flow measurement	RFC2722, for instance at http://www.faqs.org/rfcs/rfc2722.html
ipfix	IP flow export working group	http://www.ietf.org/html.charters/ipfix-charter.html
tcpdump	Packet capture and filter tool, can produce packet trace files	http://www.tcpdump.org/
GuMc99	Pankaj Gupta and Nick McKeown, "Packet Classification on Multiple Fields", Proc. Sigcomm, Computer Communication Review, vol. 29, no. 4, pp 147-60, September 1999, Harvard University	http://citeseer.nj.nec.com/gupta99packet.html

6 Appendix

6.1 List of Figures

6.2 List of Tables

6.3 Copyright Note

6.4 File Formats

The following files are read or written by netmate:

- configuration file
- meter rule file
- filter attribute and constant definition files
- meter data file

The syntax of these files is given in the following chapters.

6.4.1 Configuration File Format

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- The Document Type Definition (DTD) for the file -->

        <!ELEMENT CONFIG (MAIN ,  CONTROL , CLASSIFIER , PKTPROCESSOR , EXPORTER)>
        <!ELEMENT MAIN (PREF*)>
        <!ELEMENT CONTROL (PREF* , ACCESS)>
        <!ELEMENT ACCESS (ALLOW | DENY)*>
        <!ELEMENT ALLOW (#PCDATA)>
        <!ELEMENT DENY (#PCDATA)>
        <!ELEMENT CLASSIFIER (PREF*)>
        <!ELEMENT PKTPROCESSOR (PREF* , MODULES)>
        <!ELEMENT EXPORTER (PREF* , MODULES)>
        <!ELEMENT MODULES (MODULE*)>
        <!ELEMENT MODULE (PREF*)>
        <!ELEMENT PREF (#PCDATA)>

        <!ATTLIST MODULE
                NAME CDATA #REQUIRED
        >
        <!ATTLIST ALLOW
                TYPE (Host | User | Group) "Host"
        >
        <!ATTLIST DENY
                TYPE (Host | User | Group) "Host"
        >
        <!ATTLIST PREF
                NAME CDATA #REQUIRED
                TYPE (UInt8 | SInt8 | UInt16 | SInt16 | UInt32 | SInt32 | UInt64 | SInt64 |
                      Float32 | Float64 | Bool | String | IPAddr) "String"
   >
```

6.4.2 Meter Rule File Format

```
<?xml version ="1.0" encoding="UTF-8"?>
<!-- The Document Type Definition (DTD) for the file -->

<!ELEMENT RULESET (GLOBAL, RULE*)>
<!ELEMENT GLOBAL (PREF|ACTION|EXPORT)*>
<!-- global configuration is overruled by per rule configuration -->
<!ELEMENT RULE (PREF|FILTER|ACTION|EXPORT)*>
<!ATTLIST RULESET
          ID CDATA #REQUIRED>
<!ATTLIST RULE
        ID CDATA #REQUIRED>
<!ELEMENT PREF (#PCDATA)>
<!ATTLIST PREF
        NAME CDATA #REQUIRED
        TYPE
(UInt8|SInt8|UInt16|SInt16|UInt32|SInt32|UInt64|SInt64|Bool|Binary|String|IPAddr|IP6Addr)
"String">
<!ELEMENT FILTER (#PCDATA)>
<!ATTLIST FILTER
```

```
        NAME CDATA #REQUIRED
        MASK CDATA "0xFF"
>
<!ELEMENT ACTION (PREF*)>
<!ATTLIST ACTION
        NAME CDATA #REQUIRED>
<!ELEMENT EXPORT (PREF*)>
<!ATTLIST EXPORT
        NAME CDATA #REQUIRED
>
```

6.4.3 Filter Attribute and Constant Definition Files

```
<?xml version ="1.0" encoding="UTF-8"?>
<!-- The Document Type Definition (DTD) for the file -->

<!ELEMENT FILTERDEF (DEF | COND)*>
<!ELEMENT DEF (#PCDATA)>
<!ATTLIST DEF
REFER (MAC | IP | TRANS | DATA) #REQUIRED
OFFSET CDATA #REQUIRED
MASK CDATA "0xFF"
LENGTH CDATA "1"
TYPE (UInt8|SInt8|UInt16|SInt16|UInt32|SInt32|UInt64|SInt64|Bool|Binary|String|IPAddr|IP6Addr)
#REQUIRED
REV CDATA ""
>
<!ELEMENT COND (DEF*)>
<!ATTLIST COND
        NAME CDATA #REQUIRED
        MASK CDATA "0xFF"
        VAL CDATA #REQUIRED
>
```

6.4.4 Filter Value File Format

```
<?xml version ="1.0" encoding="UTF-8"?>
<!-- The Document Type Definition (DTD) for the file -->

<!ELEMENT FILTERVAL (DEF)*>
<!ELEMENT DEF (#PCDATA)>
<!ATTLIST DEF
NAME CDATA #REQUIRED
TYPE (UInt8|SInt8|UInt16|SInt16|UInt32|SInt32|UInt64|SInt64|Bool|Binary|String|IPAddr|IP6Addr)
"String"
>
```

6.4.5 Meter Data File Format

This file is created and written by the Meter Data Collector. In the first release only a flat file format (human-readable ASCII) will be supported. The flat file format can be used easily with simple scripts. In the future a separate export module for direct SQL storage into a mysql database will provide a more sophisticated data output. The flat file format is based on ASCII with a simple structure similar to log files. The meter should also support a direct local flat file creation (without having a Meter Data Collector) for simple setups and test purposes.

6.4.5.1 Flat File Format

The meter data varies with the action(s) specified as part of the meter rule. On the other hand the overhead should be minimized which precludes a fully self describing format. Two different export modes exist: single file and multi-file. In multi-file mode for each action used a different file is used for exporting the measurement data. The specified file name is expanded by appending the action name. In single-file mode all measurement data is exported

to a single file. The data is written to file in a table-like output. First a header line is generated for each packet processing module used by the exporting rule. This header line contains the task identifier of the rule, the current timestamp, the number of data rows and columns and the names of the data columns. After the header line one or more lines with data follow. A single empty line follows the data which is followed by another module header and data etc.:

```
task: <task-id>, module: <module-name>, tstamp: <tstamp>, rows: <rownum>, columns: <colnum>,
              colnames: attribute1, attribute2, …, attributeN
value1, value2, …, valueN
…
…
```

Example result file produced by the "text_file" data export module and netmate running a rulset with the tasks "20", "21", "24", and "25":

```
task: 25, module: count, tstamp: Nov 26 18:50:20.457787, rows: 1, columns: 6, colnames:
packets volume first_time first_time_us last_time last_time_us
  10, 889, 1069869010, 434696, 1069869019, 429230

task: 24, module: count, tstamp: Nov 26 18:50:20.457871, rows: 3, columns: 8, colnames:
packets volume first_time first_time_us last_time last_time_us packet_rate bandwidth
  1, 146, 1069869010, 434696, 1069869010, 434696, 0, 48
  4, 222, 1069869013, 690135, 1069869014, 931969, 1, 74
  5, 521, 1069869019, 426727, 1069869019, 429230, 1, 173

task: 20, module: count, tstamp: Nov 26 18:50:30.453625, rows: 3, columns: 8, colnames:
packets volume first_time first_time_us last_time last_time_us packet_rate bandwidth
  3, 236, 1069869021, 39973, 1069869021, 589580, 1, 78
  3, 180, 1069869022, 533141, 1069869024, 536102, 1, 60
  5, 790, 1069869026, 536977, 1069869027, 777747, 1, 263

task: 20, module: show_ascii, tstamp: Nov 26 18:50:30.453724, rows: 4, columns: 12, colnames:
ipversion ttl pktlen iphdrlen proto tcpudphdrlen srcip srcport dstip dstport tcpseqno tcpflags
  4, 64, 180, 20, 17, 8, 10.10.10.1070, 800, 5.5.5.5, 2049, 0,
  4, 29, 140, 20, 17, 8, 5.5.5.5, 2049, 10.10.10.1070, 800, 0,
  4, 64, 184, 20, 17, 8, 10.10.10.1070, 800, 5.5.5.5, 2049, 0,
  4, 29, 148, 20, 17, 8, 5.5.5.5, 2049, 10.10.10.1070, 800, 0,

task: 21, module: count, tstamp: Nov 26 18:50:30.454915, rows: 3, columns: 8, colnames:
packets volume first_time first_time_us last_time last_time_us packet_rate bandwidth
  3, 236, 1069869021, 39973, 1069869021, 589580, 1, 78
  3, 180, 1069869022, 533141, 1069869024, 536102, 1, 60
  5, 790, 1069869026, 536977, 1069869027, 777747, 1, 263

task: 20, module: testmod, tstamp: Nov 26 18:50:30.454575, rows: 1, columns: 6, colnames:
a_char int8_1 int8_2 int16 int32 int64
  a, -1, -125, -16000, -2000100200, -10200300400

task: 20, module: testmod, tstamp: Nov 26 18:50:30.454575, rows: 5, columns: 4, colnames: col1
col2 col3 col4
  0, 0, 0, 42
  1, 2, 10, 42
  2, 4, 20, 42
  3, 6, 30, 42
  4, 8, 40, 42

task: 20, module: testmod, tstamp: Nov 26 18:50:30.454575, rows: 1, columns: 6, colnames:
uint8 uint16 uint32 uint64 string a_binary
  250, 32000, 4000100200, 10200300400, test,

task: 20, module: testmod, tstamp: Nov 26 18:50:30.454575, rows: 3, columns: 2, colnames:
an_id a_text
  1, abcd
  2, efgh
  3, ijkl
```