***Advanced architecture for INTER-domain quality of service MONitoring, modelling and visualisation***

***INTERMON-IST-2001-34123***

# *INTERMON Deliverable 8*
# "Prototype of inter-domain QoS monitoring and structure discovery components"

| | |
|---|---|
| **Work-package No. / Title** | WP 4<br>Prototype of inter-domain QoS monitoring and structure discovery components |
| **Deliverable Type** | Public |
| Planned Issuing Date | 2003-03-31 |
| Distribution | INTERMON Consortium |
| Document Identifier | IM-WP403-v103-NEC-DS-D8 |
| Filename | IM-WP403-v103-NEC_DS_D8.doc |
| Version | v 1.03 |
| Editor/Author | *NEC Europe Ltd.* |
| Contact Person(s) | *Sandra Tartarelli <Sandra.Tartarelli@ccrle.nec.de>* |


| **AUTHOR(S)** | WP4 partner | |
|---|---|---|
| | [NEC-E] | S. Tartarelli, M. Molina, F. Raspall |
| | [SR-ANC] | I. Miloucheva, T. Pfeiffenberger |
| | [TILAB] | M. Beoni |
| | [TID] | P. A. Aranda Gutiérrez |

## Change History

| | |
|---|---|
| **V100** | Fist proposal for table of contents (NEC) |
| **V101** | Revised table of contents based on reviewers comments during the audit (NEC) |
| **V102** | Integrated contributions from NEC, SR, TILAB, TID and executive summary. Revision required. |
| **V103** | Final version |

# Table of Contents

EXECUTIVE SUMMARY ..... 5
1 IPFIX FLOW METER ..... 6
1.1 Functionality ..... 6
1.2 INTERMON use cases ..... 6
1.2.1 Volume-based charging ..... 6
1.2.2 Short term Traffic Engineering ..... 6
1.2.3 Long term Network Planning ..... 6
1.2.4 QoS management ..... 6
1.3 Design issues ..... 6
1.4 Implementation issues ..... 7
1.4.1 Meter component ..... 7
1.4.2 Collector component ..... 9
1.4.3 Planned enhancements ..... 9
1.5 Short user manual ..... 9
1.5.1 Installation ..... 9
1.5.2 Running the meter ..... 9
1.5.3 Syntax of a task specification file ..... 10
2 CM TOOLSET IN INTERMON (CM-I TOOLSET) ..... 12
2.1 Functionality ..... 12
2.2 INTERMON use cases ..... 12
2.3 Origin of the tool ..... 12
2.4 Design issues and interfaces ..... 13
2.5 Implementation issues ..... 13
2.5.1 Measurement Management Station ..... 13
2.5.2 Measurement Client Stations ..... 14
2.5.3 Installation Procedures for the Measurement Management Station ..... 14
2.5.4 Installation Procedures for the Measurement Client Stations ..... 15
2.5.5 Uninstall ..... 16
2.6 INTERMON Extensions ..... 16
2.6.1 Extension of GUI and Distributed Measurement Agents (MAa) with Path Analysis Functions ..... 16
2.6.2 Traceroute topology data base and visualisation of path topology ..... 16
2.6.3 Network tomography application and visualisation ..... 16
2.6.4 Visualisation of QoS dependencies on topology ..... 17
2.7 Test of the new components ..... 17
2.8 Short user manual ..... 17
3 MONRES BASED DATA COLLECTOR ..... 18
3.1 Functionality ..... 18
3.1.1 Management Layer ..... 18
3.1.2 Collection layer ..... 18
3.1.3 New available INTERMON features ..... 18
3.2 INTERMON use cases ..... 19
3.2.1 Basic work case ..... 19
3.3 Origin of the tool ..... 20
3.4 Design issues and interfaces ..... 20
3.4.1 Management Layer interfaces ..... 20
3.4.2 Configuration and report messages ..... 20
3.5 Implementation issues ..... 20
3.5.1 Platform requirements ..... 20

3.5.2 Required Libraries ....21

4 INTER-DOMAIN TOPOLOGY DISCOVERY TOOL ....22

4.1 Functionality ....22
4.2 INTERMON use cases....22
4.2.1 Short term observations....22
4.2.2 Long term observations ....22
4.3 Origin of the tool ....22
4.4 Design issues and interfaces ....22
4.5 Implementation issues ....23
4.5.1 Platform requirements ....23
4.5.2 Libraries....23
4.5.3 Modifications to the original MRT 2.2.2a source ....24
4.6 Tests....24
4.6.1 Test scenarios ....24
4.6.2 Test results....24
4.7 Short user manual ....25
4.7.1 Configuring, compiling and installing MRT ....25
4.7.2 Using the significant MRT components....25

REFERENCES ....27

# Executive Summary

This deliverable offers an overview of the measurements and topology discovery components developed or extended within the INTERMON project. While D3 [1] specified the functionalities of the tools, the current document focuses mainly on implementation issues. A major effort has been directed to pointing out the innovations contributed by the project, as compared to the pre-existing situation.

As already stated in previous deliverables [1], [2], a careful selection of components has been carried out, to cover a wide variety of monitoring and topology discovery functionalities:

- Passive flow measurements mainly for flow-volume information
- Active measurements for end-to-end QoS measurements
- Troubleshooting, i.e. determining situations of overload or malfunctioning
- Autonomous System topology discovery.

The first functionality is covered by the IETF IPFIX [3] compliant flow meter, which has been completely designed and implemented within the INTERMON project. The IPFIX protocol is still in the specification phase; therefore the experience gained within the INTERMON project can have an impact for the finalisation of the specification.

Active measurements are possible by using both the CM Toolset and the MonRes tool. The former was already developed within the AQUILA project [4] but it was extended for the INTERMON project to support inter-domain path monitoring, network tomography and QoS analysis dependent on the path topology.

MonRes is a commercial tool developed by TELECOM ITALIA LAB for the managing of networks, able to report on the status of HW and SW in the network, based on both active and passive measurements. Within the INTERMON project, a management layer has been designed and implemented to interact with other INTERMON components and allow, in this way, INTERMON users to access MonRes functionalities as part of the INTERMON toolkit.

The last component reported in this deliverable is the Merit's MRT, which supports the AS (Autonomous System) topology discovery function, by detecting BGP-4 routing updates. Also in this case, the original tool is publicly available [5] but modifications were required for the integration in the INTERMON architecture.

# 1 IPFIX flow meter

The IETF IPFIX Working Group has been established with the goal of standardising the export of IP flow measurement results from a monitoring station (meter) to a collector, i.e. a recipient of the measurements results. This standardisation is necessary to achieve an acceptable level of interoperability among equipment from different vendors (e.g. the meters may be embedded into the routers, and normally a collector will receive messages from more than one meter) and to support the development of generalised flow analysis tools. The INTERMON IPFIX meter is a new tool fully developed within the INTERMON project. It consists of two main components, the “meter” itself and the “collector”, with the meter sending to the collector messages in the format defined in the IPFIX standard. These messages contain the result of flow measurement.

## 1.1 Functionality

The IPFIX meter allows measuring the volume of traffic observed for a certain flow from the beginning of the measurements until the time of the last report. Reports are sent from the meter to the collector at regular intervals. IPFIX provides a very flexible flow definition [6], allowing to perform measurements at very different levels of granularity, ranging from the total traffic flowing on a link to the single stream (for instance a single VoIP call).

## 1.2 INTERMON use cases

Flow measurements can support a variety of applications relevant for both intra- and inter-domain scenarios. In the following, we list some examples of how the INTERMON user might exploit the information retrievable from the IPFIX flow meter.

### 1.2.1 Volume-based charging

This is the most straightforward application for per-flow traffic volume measurements. In the INTERMON context, for instance, a network operator A might charge a client operator B for the actual volume of transit traffic that the operator B sends to a remote destination and crossing domain A. The natural point of the network to place the meter would be at an IXP (Internet eXchange Point). The charging might be based either on the total volume of traffic or distinguishing by class of services.

Another possible application could be for charging VoIP calls, based on the actual volume of traffic exchanged during the call.

### 1.2.2 Short term Traffic Engineering

Several traffic engineering actions require per-flow traffic volume information. For instance, it might be desirable that the percentage of high priority traffic on a (possibly inter-domain) link does not exceed a certain threshold. If this happens, part of the high priority traffic might be re-routed through a different link.

### 1.2.3 Long term Network Planning

A typical application would be the upgrade of a (possibly inter-domain) link capacity, if the average link utilisation exceeds a certain threshold.

### 1.2.4 QoS management

The current version of the meter is not particularly suited for this purpose. However, in the sequel of the project the meter will be enhanced, by integrating software for passive one-way delay measurements. In this case, a straightforward application would be for measuring the delay experienced by a certain flow and support in this way SLA fulfillment verification.

## 1.3 Design issues

The passive metering tool is a probe that has access to the traffic flowing in a network. It can be located inside a network or at the boundary of a domain, e.g. in an IXP. The meter

can receive requests from a meter manager, which in turn will receive requests in XML format. Such requests may be originated by INTERMON users, and will define the flows to be measured, the observation point and the measurement interval. The results of the metering tasks are flow records, which report for each observed flow the volume of traffic (in bytes and packets) observed from the beginning of the measurements until the last reporting time. In addition, each flow report is associated to a user_ID, to identify the user who started the measurement for the specific flow and a task_ID. This information is required for storing in the INTERMON database and subsequent retrieval of the results by the user.

The flow records are exported from the meter to a data collector using the IPFIX protocol. Finally, the interface between the data collector and the remote INTERMON database is XML-based.

Figure 1 shows the IPFIX meter/collector architecture and a possible location in an inter-domain scenario.

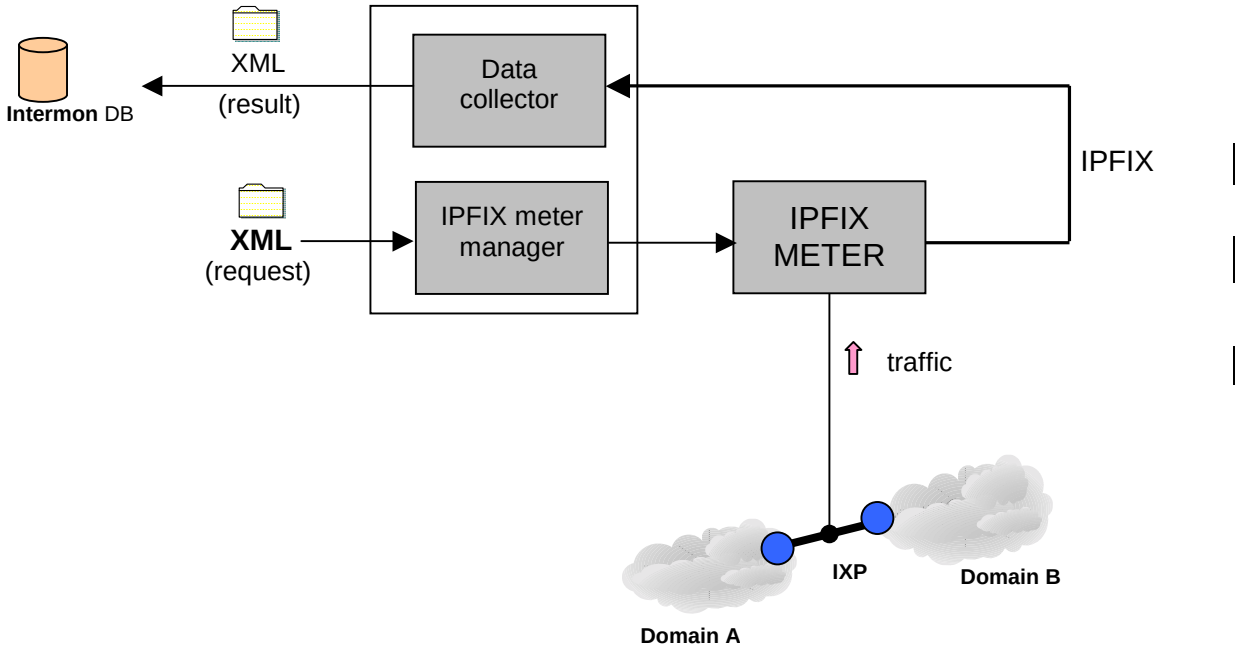


Figure 1 IPFIX meter architecture

## 1.4 Implementation issues

### 1.4.1 Meter component

The metering component has been implemented in a Linux PC platform. It has been designed as a four-stage chain of functional blocks that form a user-space process, as shown in Figure 2. Since the meter is intended to act as a probe, access to traffic is granted by means of *libpcap*, a widely used packet-sniffing library used for example by *tcpdump*. The first functional block activated when a packet is detected is the *packet analysis* block. This block extracts several fields of the network and transport level headers and stores them in a key. This key, together with the length of the observed packet and its timestamp are used as input for the following functional block: the *rule matching*. This second block has access to the data structures that store the information regarding the currently active rules, that is, the metering requests, which exactly specify what traffic is to be accounted. As an example, a rule may be configured to account all TCP over IPv4 traffic flowing from network 34.28.0.0 to network 67.23.0.0, with a specific TOS. The *rule-matching* block decides whether a packet matches any/some of the rules within the current set of rules and if so, generates a key for every rule that was matched. That is, a packet matching a rule is attributed to a flow. The key, along with the size and timestamp of the packet, is sent to the *flow cache* block.

The set of rules can be updated any time upon the reception of new metering tasks from the *passive meter manager*.

The *flow cache* block is the local data structure where information regarding the observed flows is kept. The exact position of every record in the cache to be updated by the arrival of a newly received packet is determined by means of a hash function of the key. Records that have not experienced any update for a certain time, that is, corresponding to flows without activity, are reported a last time and then deleted from the flow cache.

**A scalability remark**

The IPFIX meter has been designed to scale with high-speed links, where both packet rate and number of flows might be very high.

The above mentioned hash function has been selected to quickly associate the corresponding flow record for each incoming packet that passed the filtering phase. This solution, based on an efficient hash function, aims at achieving scalability with respect to the packet rate.

If the number of flows observed grows large, it can potentially exceed the flow cache size. In order to protect the meter against malfunctioning due to the large number of flows, we implemented a mechanism that, if enabled, allows maximising the amount of accounted traffic, by selecting those flows that mostly contribute to the total throughput on a certain link. This solution derives from the observation that in the Internet the flow size distribution is “heavy tailed”, with few “elephant” flows accounting for a good portion of the overall traffic, and a large number of “mice” flows with negligible contribution [7] and references therein.

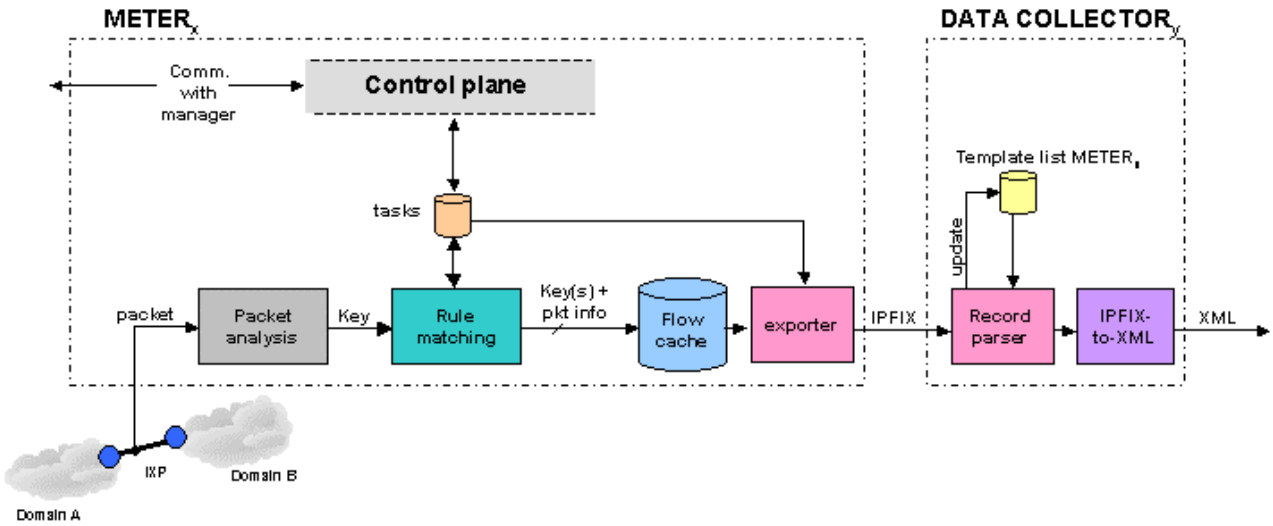


Figure 2 IPFIX meter functional blocks

The fourth block is the *exporter*. This block is responsible for the transmission of the information regarding the observed traffic (i.e. the flow records) to a remote *data collector.* The reporting period can be configured, but in the current implementation it has to be the same for all flows. The export protocol used between the meter and the data collector is IPFIX-compliant. The implemented protocol follows CISCO NetFlow v9, which is the protocol chosen by the IETF IPFIX WG. However, differently from the CISCO NetFlow tool, which uses UDP as the transport protocol, the IPFIX IETF WG has recommended the use of TCP as the transport protocol beneath IPFIX. This is also the transport protocol chosen for the INTERMON IPFIX meter implementation, in order to grant the reliability and the congestion control features required by the future IPFIX standard. The meter, therefore, acts as a TCP client that establishes a TCP connection with a remote *data collector*. The latter can receive reports from several meters.

Note that differently from the other three functional blocks, which can potentially be activated upon arrival of every packet, the exporter is activated only at regular reporting intervals. However, needing to read all the flow cache to prepare the IPFIX messages with the measurement results, it will generate at regular intervals “bursts” of CPU utilization.

### 1.4.2 Collector component

The collector is implemented as a server receiving on different TCP connections (one per each contacting meter) flow measurement records in IPFIX format.

Its main role is to parse the IPFIX packets and to produce XML data in the format to be written in the INTERMON database, as shown in Figure 1.

The main implementation remark regards the above mentioned choice to use TCP as the underlying transport protocol between the meter and the collector. In this respect, we found that one of the fields of the export packet header of CISCO NetFlow v9 (originally designed to use UDP) carries a packet sequence number. If TCP is used, having a packet sequence number is useless as TCP will take care of re-transmitting potentially lost packets. On the contrary, the information about the length of the export packet (in bytes) is missing and this would be extremely useful to ease the collector’s job in rebuilding export packets fragmented in different TCP segments. So, we decided to replace the above mentioned 32 bit sequence number field with the export packet length. This proposal will be carried to the IPFIX working group.

### 1.4.3 Planned enhancements

In the sequel of the project, some enhancements to the tool are planned. In the first place, we will extend the flexibility of the flow definition. Currently, each field (e.g. L4 source port) is used either for specifying a single value to be associated to a certain flow, or to perform flow detection based on that field (e.g. packets with different L4 source port values will generate separate flows). We will extend the meter, so that a flow can be also identified by a range of values (e.g. of the L4 source port) or a list of values.

The major enhancement planned so far is the integration of software to support passive one-way delay measurements. To this end, we will use meters synchronised through GPS probes.

Other enhancements (mainly performance oriented) may derive from a code profiling activity just started.

## 1.5 Short user manual

### 1.5.1 Installation

Before installing the meter, the libpcap library has to be installed in the target machine. The libpcap sniffing library can be found at http://www.tcpdump.org

The meter comes with a very simple makefile. Once the library has been installed, simply move to the parent directory of the source tree where the source code is and type make.

### 1.5.2 Running the meter

Once the meter has been compiled, it can be launched[1] with the following command line, where all the fields are mandatory, except from the first one:

**`meter [-f task.file] dev r_period IP_meter port_meter IP_coll port_coll`**

Where:

**`-f`** is a flag to tell the meter to consider the tasks specified in the file **`task.file`**

**`task.file`** a text file which lists a set of metering tasks, written with a defined syntax explained below. The reason for the existence of this option is because to allow the use of

[1] Super user rights are required to launch the meter.

the meter in standalone mode, that is without the need to be configured from an INTERMON *passive meter manager*.

**dev** is the name of the Ethernet interface of the Linux PC hosting the meter whose traffic is desired to be measured. That is, it is the name of the Ethernet interface acting as a probe, for instance “eth1”

**r_period** is the period of time in seconds between two consecutive reports.

**IP_meter** is the IPv4 address assigned to the meter. If the machine hosting the meter has two interfaces, say eth0 and eth1, then eth0 can be the one put in promiscuous mode for the observation of the traffic, while eth1 can be used for the reporting traffic. If such configuration is desired, the argument `IP_meter` has to be the IPv4 address associated to eth1.

**port_meter** is the TCP port that the meter should use for the exporting to the data collector

**IP_coll** is the IPv4 address where the remote collector can be reached.

**port_coll** is the TCP listening port where the data collector will be awaiting for connections that come from any meter.

### 1.5.3 Syntax of a task specification file

It may be convenient to run the meter in standalone mode. In such situations, a list of metering tasks can be provided creating a file and passing it as an argument with the –f flag when the meter is started.

The syntax of this file is the following (the fields in italics are values):

```
ruleid value
ver    {4,6}
start  time
stop   time

[saddr flag value mask/prefix]
Saddr filter address mask/prefix
saddr both address mask/prefix
saddr detect [mask/prefix]
[daddr flag value mask/prefix]
daddr filter address mask/prefix
daddr both address mask/prefix
daddr detect [mask/prefix]
[proto flag value]
proto filter value
proto detect
[tos flag value]
[tos filter value]
[tos detect]
[sport flag value]
[sport filter value]
[sport detect]
[dport flag value]
[dport filter value]
[dport detect]
[f_label flag value]
[f_label filter value]
[f_label detect]
[icmp_type flag value]
```

**[icmp_type filter** *value***]**
**[icmp_type detect]**

where:

**ruleid** is an integer identifier of the metering task and should be different for every task. In the context of INTERMON, its exact value may be provided by the local controller, and my be derived from a user_id and task_id so that the result of the metering data can be assigned to a specific user in the INTERMON database.

**ver** is the IP version, either 4 or 6.

**start** is the time when the task should be processed by the meter, i.e. when the measurements for the specified traffic will start to be effective.

**stop** is the time when the task should become obsolete, that is, the time beyond which no such measurement will no longer take place. The format to specify both times is that one specified in RFC2579.

The four fields above are mandatory. The meter will reject a task that does not provide that information in that order. The rest of the fields are optional and allow for more fine-grained measurements (with only the first four fields, the only macro-flows that could be measured are “all IPv4 traffic” and “all IPv6 traffic”).

For the rest of the fields, the common syntax is: ***field*** *flag value1 value2*, where the possible combinations are shown above. The flag indicator can have up to 3 different values: filter, detect and both. The meaning of the flag is the following:

***filter*** - this forces the meter to match the packet’s header field with the values specified, possibly masked by some mask or prefix. If this match is not successful, the packet will not be accounted.

***Detect*** - this forces the meter to consider the field as a feature for distinction of flows, but does not impose any condition on its value. That means that a different flow record can be created for every observed value of field in the traffic. It can be accompanied by a mask or prefix in case of network addresses. In that case, the masked value of the field is the value that is used to distinguish flows. For example, “*saddr detect 255.255.0.0*” would instruct the meter to detect IPv4 flows depending on the first two bytes of the IPv4 address.

***Both*** – only applies to addresses. This forces the meter to consider for classification all traffic that matches some address with some mask, but to generate a different flow record depending on the remaining (masked) bits.

Example:

ruleid 18090
ver 4
start 2003:4:1,12:30:0.0,+1,0
stop 2003:4:1,13:00:0.0,+1,0

saddr both 34.23.0.0 255.255.0.0
dport filter 8080
proto filter tcp

This task would instruct the meter to measure all IPv4 TCP traffic, with a destination TCP port exactly equal to 8080 from addresses 34.23.x.y and to generate a different flow record for every pair (x,y) observed.

# 2 CM Toolset in INTERMON (CM-I Toolset)

## 2.1 Functionality

The INTERMON CM Toolset (CM-I Toolset) [10], [9] extends AQUILA CM Toolset [12] with the following functions:

- Path topology monitoring based on traceroute (CM Toolset user interface and daemons)
- Traceroute topology data base and visualisation
- Network inferencing (user interface and visual output)
- QoS dependency on topology (visual output based on the extended data base evaluation)

## 2.2 INTERMON use cases

The usage cases of the INTERMON CM toolset are presented as papers at the Inter-domain performance and simulation workshop (IPS 2003).

- Path Analysis and multivariate QoS dependency on topology [9]
- Network tomography, i.e. inferencing of inter-domain path characteristics based on active end-to-end QoS monitoring – scenario studies in emulation environment [10]

## 2.3 Origin of the tool

The CM Toolset was initially developed within the AQUILA project www.ist-aquila.org.

The CM Toolset distributed measurement architecture is characterised by a modular designed to reach the following goals:

- to emulate end-user applications and to characterise the mapping of their end-to-end QoS requirements into network services in a quantitative way, i.e. measurement of performance metrics for generated single and aggregated flows;
- to maintain and store measurement information of different types (end-to-end performance, path metrics, network traffic, etc.) in order to be used for a mapping of the user end-to-end QoS to the network service requirements;
- to offer online access to measurement information, e.g. to enhance the network operation by the use of path performance metrics or packet drops in network elements.

These goals are achieved by combining the following new tools for distributed active and passive measurements interacting via using a common, shared measurement database:

- **End-to-end flow generators for single and aggregated flows** based on distributed agents for end-to-end performance measurement. The goal is the generation of synthetic traffic emulating typical end-to-end applications (e.g. VoIP, streaming audio, streaming video, WEB access, etc.). To implement these generators, **passive measurements of real applications**, which should use the network, are done to obtain traces for traffic generation and to derive traffic models. Existing models of other research activities are considered.
- **Active network probing agents** for obtaining path performance metrics (e.g. packet delay or packet loss as defined by IPPM) for characterisation of reachability / connectivity as well as performance metrics of particular paths and links in the access and core networks.

## 2.4 Design issues and interfaces

CM Toolset INTERMON is a tool that can be attached directly to the INTERMON architecture and be called as a configurable service from the INTERMON Toolkit GUI, using either the CM Toolset database or the INTERMON database.

Figure 3 shows the implementation structure of the CM Toolset with identification of components, which are updated in the CM-I Toolset release:

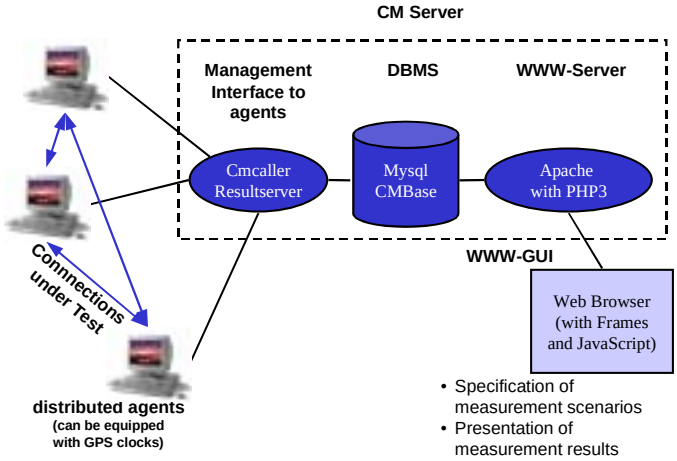


Figure 3: Implementation structure of CM Toolset

CM-Toolset implementation architecture includes the following components:

- Distributed Measurement Agents, which performs the scenarios – updated with traceroute topology scenarios in the CM-I Toolset Version.
- Mysql Cmbase updated with traceroute topology data base
- WWW GUI enhanced with visualisation for traceroute topology discovery
- CM-Caller with new functions for path monitoring
- Client with Internet browser, which is responsible for the management of the test scenarios.

## 2.5 Implementation issues

The distributed measurement system CMToolSet mainly consists of a measurement server (or measurement management station) and several measurement client stations. This section is structured as follows: first, we describe the different file-packages necessary for the Server and the Client stations, afterwards a sub-section follows, which contains the installation instructions for these packages.

A Linux (min. Kernel Version 2.4.x) standard installation is required for the installation of the software described below.

The following list contains an overview on the software to be installed on the different machines:

### 2.5.1 Measurement Management Station

#### 2.5.1.1 Basic Installation

- NTP Package The Network Time Protocol is used to synchronise the time of a computer client or server to another server or reference time source, such as a radio or satellite receiver or a modem.
- Apache Web-Server with PHP/Perl-Support, AUTH module for MySQL Database Standard apache server, provides the database access for users.
- MySQL database
- Unix ODBC driver + MySQL specific MyODBC driver

- traceroute traces the route taken by packets over a TCP/IP network
- ifconfig is used to configure the kernel-resistant network interfaces
- gnuplot

#### 2.5.1.2 CMToolSet Installation

- CMBase based on MySQL database (http://www.mysql.com) used for management of scenario information and measurement data.
- CMCaller builds the interface between the daemons and the database
- User interface (WWW-GUI)

### 2.5.2 Measurement Client Stations

#### 2.5.2.1 Basic Installation

- NTP Package
- traceroute
- ifconfig

#### 2.5.2.2 CMToolSet Installation

- CMDaemon Distributed measurement agents. They are processes running on hosts, which are used for the end-to-end measurement.

In the following the files are listed, which are needed to follow the installation instructions below. [yy.mm.dd] is for the date of the version. Be sure to take the latest version! The contents of the tar.gz-files can be displayed by using the command:"tar -tzf filename"

- **cmtoolset server basic-yy.mm.dd.tar.gz:** contains the NTP-package, Apache, PHP, MySQL, UnixODBC, traceroute, ifconfig
- **cmtoolset server-yy.mm.dd.tar.gz:** contains the binaries and start/stop scripts for CMCaller, the WWW-GUI, the NTP-package and traceroute and ifconfig
- **cmtoolset client basic-yy.mm.dd.tar.gz:** contains the NTP-package and traceroute and ifconfig
- **cmtoolset client-yy.mm.dd.tar.gz:** contains the binaries and start/stop script for the cmdaemon (measurement agent).

### 2.5.3 Installation Procedures for the Measurement Management Station

#### 2.5.3.1 Basic Installation

A single installation script for all parts of the basic software required by the CMToolSet components is provided. The Installation contains the components listed above (without Linux).

The installation script installs all necessary files using absolute path names. It is therefore recommended to use a fresh installed SuSE Linux 7.3 standard installation (if necessary with X-Server). Updated files which probably come along with bug-fixes can be installed (overwritten) separately afterwards. Furthermore a de-installation script is produced during the installation process, which can be executed to remove exactly the installed files.

Warning: Files, which are already on the machine before installation and part of the installation file tree will be overwritten.

- Login as Root
- Copy the most recent cmtoolset server basic-yy.mm.dd.tar.gz to your machine (e.g. by scp to /root/)
- extract the shell script install cmtoolset server basic from the archive and execute the script as follows:
  ```
  tar -xzf cmtoolset server basic.tar.gz install cmtoolset server
  basic-yy.mm.dd
  ```

```
./install cmtoolset server basic cmtoolset server
basic.yy.mm.dd.tar.gz
```

- Reboot the computer or start the services as described on the screen

#### 2.5.3.2 CMToolSet Installation

As for the basic installation, an Installation-script is provided. The installation contains the files listed above (without Linux).

- Copy the most recent cmtoolset server-yy.mm.dd.tar.gz to your machine (e.g. by scp to /root/)
- extract the shell script setup cmtoolset server from the archive and execute the script as follows:
```
tar -xzf cmtoolset server-yy.mm.dd.tar.gz install cmtoolset
server
./install cmtoolset server cmtoolset server-.yy.mm.dd.tar.gz
```

Reboot the computer or start the services as described on the screen.

#### 2.5.3.3 Configuration of the CMCaller

The CMCaller comes with a standard configuration file, which can be found in /etc/cmcaller.rc. The parameters that can be configured are described within this file. If the installation was done like described above, usually only the variables EAT name, EAT username and EAT password have to be adapted to configure, which EAT will be contacted by the CMCaller for resource reservations. Note that the CMCaller has to be restarted /etc/init.d/cmcaller stop; /etc/init.d/cmcaller start) after the changing the configuration file.

### 2.5.4 Installation Procedures for the Measurement Client Stations

#### 2.5.4.1 Basic Installation

As for all the other components of CMToolSet an installation-script is provided.

- Login as Root
- Copy the most recent cmtoolset client basic-yy.mm.dd.tar.gz to your machine (e.g. by scp to /root/)
- Extract the shell script install cmtoolset client basic from the archive and execute the script as follows:
```
tar -xzf cmtoolset client basic.yy.dd.mm.tar.gz install
cmtoolset client basic
./install cmtoolset client basic cmtoolset client basic-
.yy.mm.dd.tar.gz
```

Reboot the computer or start the services as described on the screen

#### 2.5.4.2 CMToolSet Installation

As for all the other components of CMToolSet an installation-script is provided.

- Login as Root
- Copy the most recent cmtoolset client-yy.mm.dd.tar.gz to your machine (e.g. by scp to /root/)
- extract the shell script install cmtoolset client from the archive and execute the script as follows:

```
tar -xzf cmtoolset client-yy.mm.dd install cmtoolset client
./install cmtoolset client cmtoolset client-.yy.mm.dd.tar.gz
```

- Reboot the computer or start the services as described on the screen

Change the RESULTSERVER variable (default is 10.0.1.1, which is the CMS in Vienna) by editing the cmdaemon startscript. (/etc/init.d/cmdaemon) to your CMCaller IP address (measurement management station). Be sure that the CMCaller is up and running, when the daemons will be started, so that they can register themselves at the management station.

#### 2.5.4.3 Start and Test

The CMDaemon can be controlled by executing `/etc/init.d/cmdaemon start|stop|status`. After the starting the CMDaemon (`/etc/init.d/cmdaemon start`), some start log messages, like the version info, GPS location info (if available) and status of start-up will be displayed on the terminal (stdout). The logging messages for errors and events of the CMDaemon are in `/var/log/cmdaemon.log` They can be checked e.g. with `tail -f` after start, i.e.:

```
/etc/init.d/cmdaemon start
tail -f /var/log/cmdaemon.log
```

The number of logging messages can be configured within the cmdaemon startscript (`/etc/init.d/cmdaemon`) by changing the variable “debug”.

With `/etc/init.d/cmdaemon status` on can get all running CMDaemon processes.

#### 2.5.4.4 Update

To update the CMDaemon download the newest version of the cmtoolset client-yy.mm.dd.tar.gz file, reinstall and restart the CMDaemon like described above.

### 2.5.5 Uninstall

Simply execute the de-install-script (located in the same directory from where you extracted cmtoolset clientyy.mm.dd.tar.gz) to delete the files related to CMDaemon.

## 2.6 INTERMON Extensions

### 2.6.1 Extension of GUI and Distributed Measurement Agents (MAa) with Path Analysis Functions

The topology monitoring scenarios for the end-to-end communication path based on traceroute are specified via user interface.

The user can set parameter for the execution of the traceroute call - monitoring interval and the type of communication service (DiffServ class).

The measurement agents for application-like measurement flows (CMDaemons) are distributed along the network, which should be measured. They retrieve the parameters for their measurement flows and also report the measurements results back to the management process (CMCaller). The agent functions are extended with PathAnalyser functions for executing of traceroute from source to destination with specific monitoring interval and type of communication service as specified by the user.

### 2.6.2 Traceroute topology data base and visualisation of path topology

The CM Toolset data base is a MySQL data base which is extended in the framework of INTERMON with following elements

- traceroute scenario description (stores the parameters of the traceroute topology scenario set by the user such as monitoring interval and type of service)
- traceroute call output
- RTT measurements done in each traceroute call.

The visualisation of path topology is for given time interval – only the changed path topologies are visualised with the time indication when the topology change is performed.

### 2.6.3 Network tomography application and visualisation

The CM-I Toolset GUI includes additional menu for network tomography visualisation. Network tomography means to inference the QoS parameter behaviour of a flow, which cannot be directly measured from the QoS behaviour of directly measured flows.

The CM-I toolset tomography is based on obtaining of the QoS parameter behaviour of a flow contained in the spatial composition of end-to-end flow.

The input parameters of the application are QoS parameter data measured for the end-to-flow and QoS parameter measured data for the spatial flow. The QoS parameter data for the "hidden" spatial flow of the end-to-end flow are obtained and visualised. Multivariate QoS parameter visualisation is provided.

### 2.6.4 Visualisation of QoS dependencies on topology

The CM-I Toolset is enhanced with functions to indicate topology changes in the QoS parameter behaviour. This function is based on the comparing of traceroute scenarios and QoS measurement scenarios and results data stored in the CM Toolset database.

The two time series data sets are compared and the topology changes are indicated in the visualisation of the QoS parameter behaviour sheets.

## 2.7 Test of the new components

The new components are currently under test. Figure 4 depicts the components of the test environment:

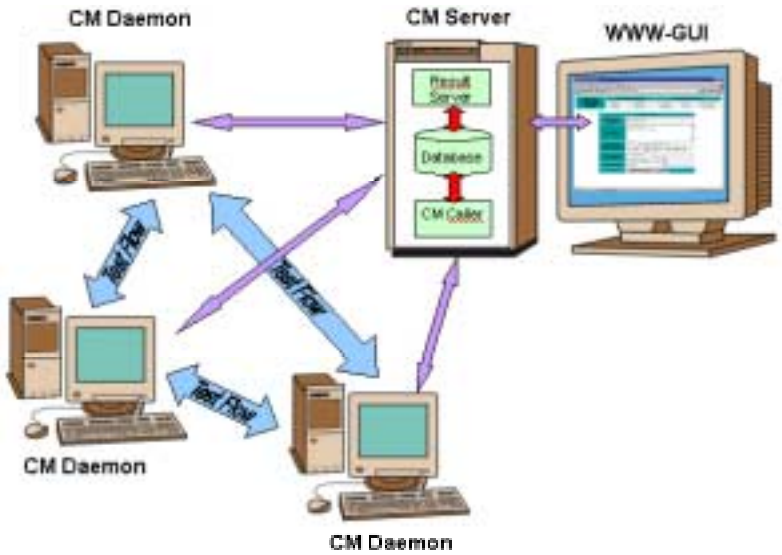


Figure 4: Test environment for CM-I Toolset component test

The testing environment includes optionally the NIST Net network emulation package developed at National Institute of Standards and Technology. NIST Net allows a single Linux PC set up as a router to emulate a wide variety of network conditions.

The end to end delay is measured between Workstations with installed CM daemons, which are synchronized directly to an GPS synchronised host or by NTP.

First results are published [9], [10].

## 2.8 Short user manual

The user manual on configuration of the tool will be supplied in INTERMON Web pages.

# 3 MonRes based Data Collector

## 3.1 Functionality

The MonRes-based tool is a network data collector performing both active (implementing a Cisco SAA probe) and passive (polling the MIB tree of managed devices) measurements. The new tool basically has two layers architecture:

### 3.1.1 Management Layer

The Management Layer performs message exchange with the INTERMON Tool Manager and drives all the existing probes in the level below.

### 3.1.2 Collection layer

It contains the network probes to interact with network hosts. The collection layer is composed of three main modules:

- **Device Interface discovery.** Interface status and name have to be known by the tool manager. This feature allows the tool management layer and the TM to know the device configuration information of the managed hosts in order to define all the active resources to collect. The module supports the discovery of different vendor interfaces.
- **Cisco SAA probe driver** is a management tool that implements Cisco router Service Assurance Agent. The probe stores, clears, starts, stops and reads the user defined measures. Currently ICMP, UDP, JITTER and HTTP measures are implemented.
- **Tilab SNMP-MIB poller** that collects both pre-defined traffic variables for each known interface and generic user-defined MIB variables. The supported polling granularity at this stage is
    - **Real time** (1 sample per minute)
    - **15 minutes**
    - **30 minutes**

The collection layer is extensible with other network probes integrating and adapting other MonRes modules (e.g. BMPOP: a tool simulating Dial-Up client connection) or other customised collecting probes.

### 3.1.3 New available INTERMON features

The new collection layer probes are an integration of MonRes hard coded functions to a new independent user-friendly Unix command line tool. Therefore the tool management layer interacts with probes with this command line with the aid of some configuration files and defines a message and configuration interface. Tool Manager can easily configure measurements campaigns simply using functions the interface provides.

The SAA probe has been re-developed in order to override these hard coding problems starting from the previously router behaviour analysis done during the MonRes integration phase.

The JITTER measure type has been integrated in the SAA probe during the INTERMON measurement toolkit develop. This measure simulates phone calls so that is possible to retrieve delay variance.

## 3.2 INTERMON use cases

### 3.2.1 Basic work case

#### 3.2.1.1 Static information discovery

This information is used for configuring campaign, by specifying the interface resources the user wants to collect and to compute indicators (such as throughput) using the interface bandwidth.

TM has simply to request information for a list of known devices specifying the address and SNMP community.

#### 3.2.1.2 Discovery results

For each requested node the following are the info the tool replies:

- Interface Name
- Interface Type
- SNMP management index
- Bandwidth

#### 3.2.1.3 Configuration Campaign

TM defines a user defined campaign specifying for each device the interfaces to be collected and the distributed SAA measures to be configured in order to retrieve delay (or download time) to peers and peers sub-networks.

#### 3.2.1.4 Campaign changes

TM can change a previously configured campaign by adding or deleting both devices or configured device measures.

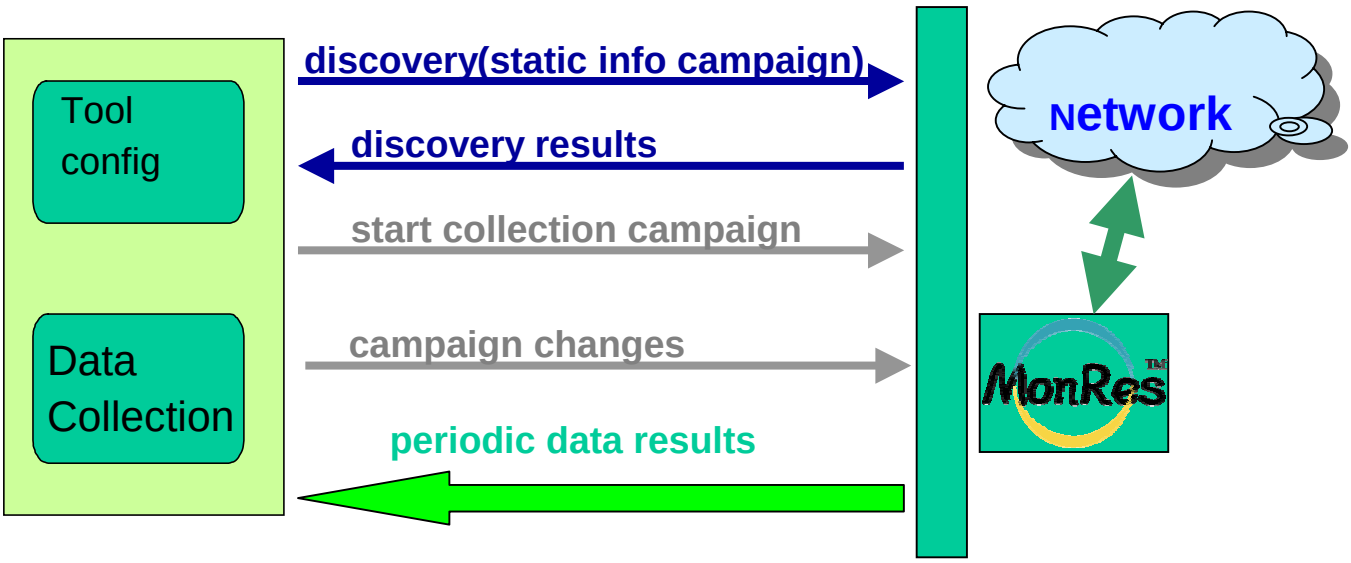


Figure 5 – basic message exchange

#### 3.2.1.5 Periodic data results

The management layer sends out, according to a polling period, to the TM the out variables defined for each type of measure, providing also an interface to store it in the LC INTERMON database.

## 3.3 Origin of the tool

The tool, as cited before, has been developed using functions of the Tilab network management-monitoring platform MonRes™. The platform has three layers: data collection; database management and data aggregation / report generation. This platform is a product that is adapted to different Telecom Italia network scenarios with reliability and load efficiency. The system uses a distribute socket communication (IPC) to transfer and to store data to the binary proprietary database. The user interface and the visualization are performed by a web based CGI application. Software is mainly C code and shell script. The main shortcoming is that most of the components are hard-coded in the whole architecture. This choice was taken to improve processing and collection performance due to bulk data involved. The platform web page is http://www.monres.com.

## 3.4 Design issues and interfaces

### 3.4.1 Management Layer interfaces

The main Management layer role is to provide an interface to the INTERMON TM to define, modify, start and stop collection rules independently of the probes located in the layer below.

### 3.4.2 Configuration and report messages

Basic configuration message system is based on an XML interface that has been defined for all available measures. TM performs a configuration request campaign sending an XML file specifying for each node the resource to collect.

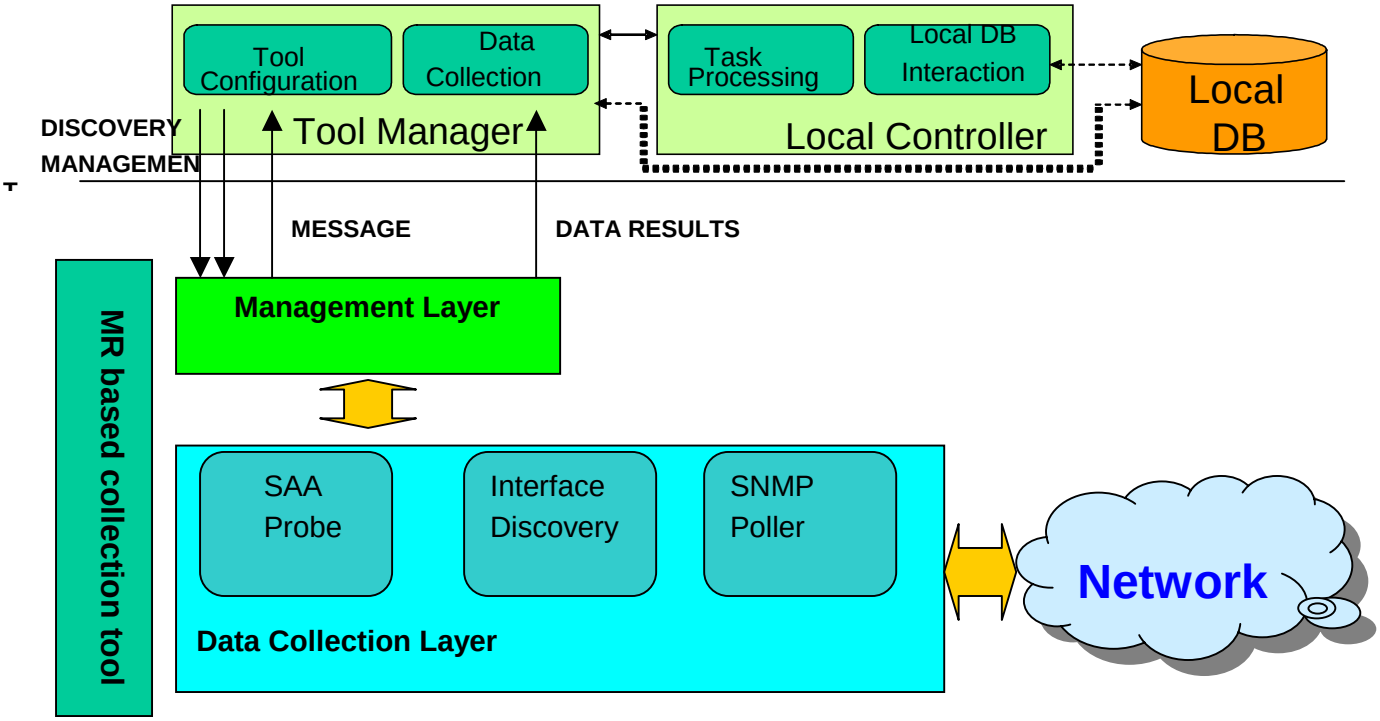


Figure 6

MonRes management layer system sends out to LC reports in XML format.

The interface that helps storing collected data into the Local DB is not fully defined yet.

## 3.5 Implementation issues

### 3.5.1 Platform requirements

- Unix/Linux environment.

### 3.5.2 Required Libraries

- Net-SNMP
- JAXP 1.2

# 4 Inter-domain Topology Discovery Tool

## 4.1 Functionality

The topology discovery tool includes the command interface, the toolkit and sources of inter-domain routing information.

The command interface to the INTERMON toolkit is implemented in a tool wrapper which:

- receives the control commands via an XML-RPC based communication; and
- returns the routing data as XML records using the same XML-RPC communication channel.

The inter-domain routing information sources include:

- **the provider's routing infrastructure**, in which case a BGP-4 speaker is used which implements the communications protocol defined in BGP-4 but does not inject routing information.
- **Internet routing information repositories,** in which case files with the routing information in binary format are used to feed the INTERMON database

## 4.2 INTERMON use cases

### 4.2.1 Short term observations

Based on live routing information obtained from one or more BGP-4 routing infrastructures. Objective is to detect routing updates, which might have an impact on the routing paths.

### 4.2.2 Long term observations

Based on live routing information obtained from one or more BGP-4 routing infrastructures or information coming from the Internet routing information repositories. Objective is to reconstruct the topological information on a specific part of the network, calculate the topology's evolution and detect the updates, which had an impact on the routing paths.

## 4.3 Origin of the tool

The topology discovery tool is based around the Multithreaded Routing Toolkit provided by Merit. The toolkit as such is available in the Internet. The source code CVS tree hosted by the **sourceforge** project. The project home page is http://www.mrtd.net

## 4.4 Design issues and interfaces

Figure 7 shows the tool architecture:

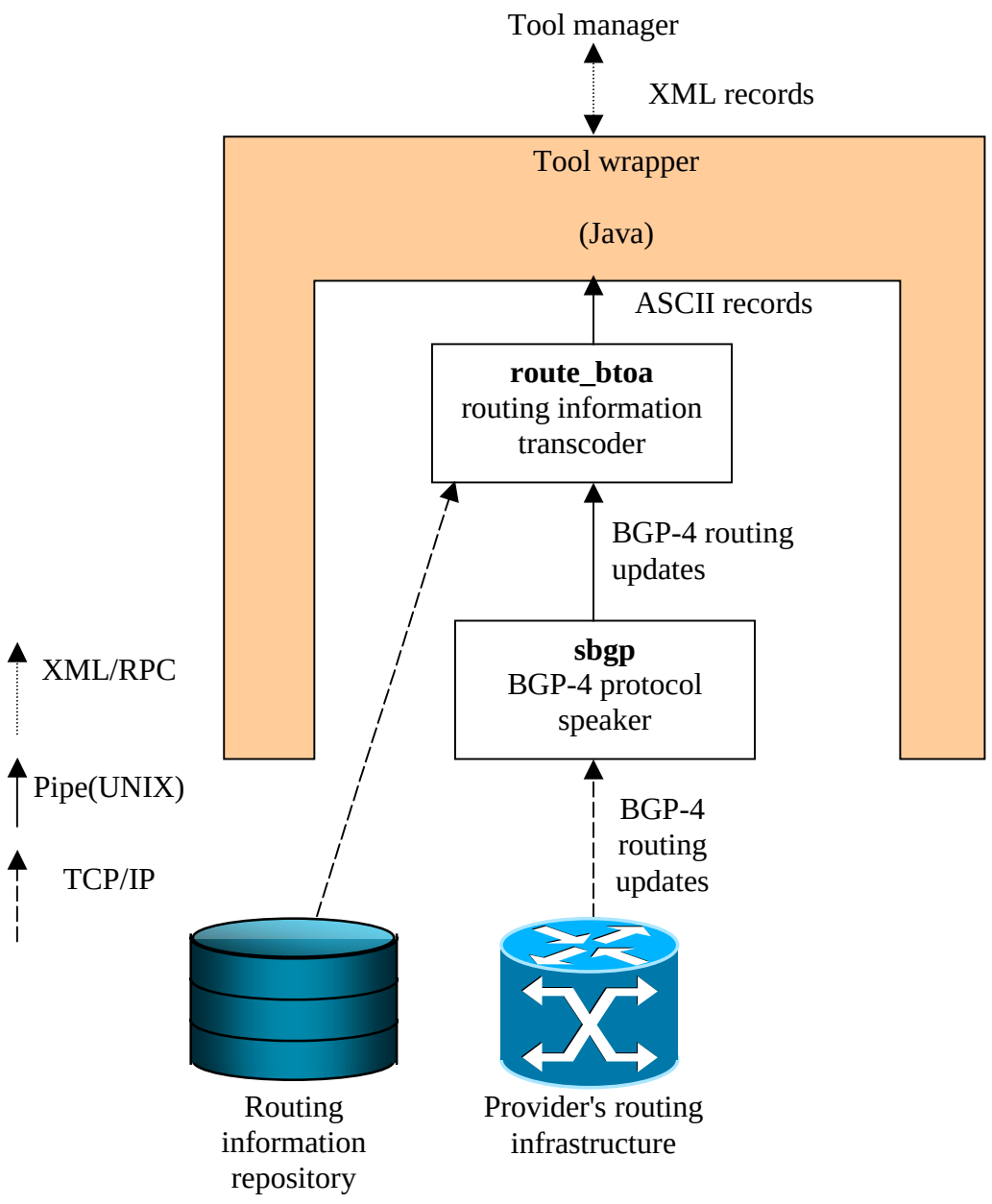


Figure 7 Tool architecture

The tool wrapper is a software component written in Java. It implements the following functions:

- Receiving the commands from the local controller via an XML-RPC interface
- Controlling the tools; i.e. starting and stopping the MRT programs
- Retrieving data output from the MRT programs and translating it to XML records which can be stored in the databases.

## 4.5 Implementation issues

### 4.5.1 Platform requirements

- Linux with 2.4.x kernel for Merit's MRT
- J2SDK 1.4 for the tool wrapper

### 4.5.2 Libraries

- None

### 4.5.3 Modifications to the original MRT 2.2.2a source

The original MRT 2.2.2a source has shown the following several functional shortcomings during the unitary tests and first integration effort for the project review demo:

1. **route_btoa**, the filter which converts the binary routing information to ASCII bails out with an exception and core dump when it finds an update in the binary routing information produced by the death of one of a peer. The routing record will have invalid peer information (NULL).
2. **route_btoa** will write to the standard output file and make use of the operating system's buffer handling. This results in the BGP wrapper reading in blocks of 4 K bytes of ASCII. These blocks might begin or end by incomplete ASCII routing records, which require extra processing in the BGP wrapper. It also causes high retransmission delays for most records, which are unacceptable at the normal rates in which updates appear in a routing infrastructure.
3. **sbgp** can be executed by any user, but it will only be able to open the BGP-4 socket when executed with root privileges. Any other user will block the use of the TCP port used by BGP-4, effectively blocking the topology probe without the BGP wrapper ever taking notice of this fact.

The source needs to be patched with the patch file **wp4/topology/unix/mrt-2.2.2a.diff** from the CVS server to overcome the above shortcomings.

## 4.6 Tests

### 4.6.1 Test scenarios

The following tests have been carried out:

1. establishing BGP-4 sessions from sbgp
    a. to a router with BGP-4 routing enabled using a PC with Linux and zebra configured as a BGP-4 router
    b. to the BGP-4 simulation program **bgpsim** included in MRT
2. binary to ASCII conversion of BGP-4
    a. Using files coming from Internet routing update repositories
    b. Using the input from the simple BGP-4 speaker with a session established to a router (see 1.a)
    c. Using the input from the simple BGP-4 speaker to the BGP-4 network simulator program **bgpsim** included in MRT (see 1.b)

### 4.6.2 Test results

The following table shows the status

| Test | Result | Reason | Corrective actions |
|---|---|---|---|
| 1a | Passed | N/A | N/A |
| 1b | Passed | N/A | N/A |
| 2a | Failed | shortcoming 4.5.3(1) | Patch which identifies routing records which produce exception and prevents offending code to process them |
| 2b | Failed | shortcoming 4.5.3(1) | See above |
| 2c | Passed | N/A | N/A |

## 4.7 Short user manual

### 4.7.1 Configuring, compiling and installing MRT

The following steps have to be taken to successfully compile and install the topology detection tools. Make sure to log in as root.

1. Download and unpack the source files for Merit's Multithreaded Routing Toolkit.
2. Apply the patch which is available on the CVS server
3. ***Ignore*** the `make-sym-links` script on the base diretory; directly move to the directory `mrt-2.2.2a/src`
4. Execute the configuration script:
   `./configure –-disable-ipv6 –-disable-mrouting –-disable-qrouting`
5. Edit the file Make.include and include the flag `–DNDEBUG` to `CPPFLAGS`
6. Check that the configuration script set all the relevant definitions the file `include/config.h` correctly and modify this file if necessary.
7. Run
   `make depend`
   and ignore all messages.
8. Run
   `make all`
   and
   `make install` after the tools have been successfully compiled.

### 4.7.2 Using the significant MRT components

#### 4.7.2.1 Building and using the MRT topology probe

The MRT topology probe is constructed via a UNIX shell command, which calls **sbgp** and **route_btoa** with the correct parameters. In a normal environment, an internal BGP4 session has to be established between a significant router in the provider's routing infrastructure and the topology probe. The best choice would be a router, which sits in a route reflection hierarchy. This insures that the topology probe gets all the routing updates from the provider.

The topology probe script is:

```
#!/bin/sh
#     $1 : the providers autonomous system in 'Asxxx' format
#           where xx is the AS's id
#     $2 : the IP address of the BGP-4 peer sending the routing updates
sbgp $1 $2 $1 –o stdout | route_btoa –m –i stdin
```

The topology probe script prints the routing records in ASCII format. These routing records have to be parsed and converted to XML by the BGP wrapper

#### 4.7.2.2 Using MRT to parse files from Internet routing repositories

**route_btoa** can also be used to parse files from Internet routing repositories. The command would be

```
route_btoa –m –i <file from routing repository>
```

#### 4.7.2.3 Using BGPSIM to debug the topology tools

The bgp simulation program **bgpsim** can be used to recreate specific network situations during the SW integration and debugging phases. The program is driven via a configuration file with CISCO-alike syntax and simulates a router sending stable and unstable routes to the topology probe.

**bgpsim** is called with the command

```
bgpsim –v f <configuration file>
```

The following listing shows all following test cases:

1. Stable routes, which are only announced once
2. Flapping routes, which announced and withdrawn
3. Flapping routes, which are announced with varying attributes

```
 password foo
!
! Test stable network advertisement
!
network-list 1
  range 10.168.0.0/24 10.168.13.0
  map 1
!
route-map 1
  set next-hop 198.108.60.20
  set as-path 10
  set metric 3
  set origin igp
!
! Test route flapping between two providers
!
network-list 5
  range 10.168.16.0/24 10.168.21.0
  map 50 51
  change 20
!
route-map 50
  set next-hop 198.108.60.20
  set as-path 10 200 2000
  set metric 30
  set origin igp
!
route-map 51
  set next-hop 198.108.60.30
  set as-path 10 201 2000
  set metric 3000
  set origin igp
!
! Test route flapping from one provider
!
network-list 6
  range 10.168.22.0/24 10.168.31.0
  map 50
  stability 30 jitter 5
!
route-map 50
  set next-hop 198.108.60.60
  set as-path 20 200 3000
  set metric 30
  set origin igp
!
! Configure BGP-4 session
! 65001           is AS identifier
! 192.168.10.61   is IP address of BGP-4 peer
router bgp 65001
  neighbor 192.168.10.61 remote-as 65001
```

# REFERENCES

[1] INTERMON Deliverable 2, System Architecture and Specifications, INTERMON partners, August 2002.

[2] INTERMON Deliverable 3, Specification of inter-domain QoS monitoring and structure discovery component, INTERMON partners, September 2002.

[3] http://www.ietf.org/html.charters/ipfix-charter.html

[4] http://www.ist-aquila.org/

[5] http://www.mrtd.net

[6] J. Quittek, T. Zseby, B. Claise, and S. Zander, Requirements for IP Flow Information Export draft-ietf-ipfix-reqs-09.txt, February 2003.

[7] M. Molina, A Scalable and Efficient Methodology for Flow Monitoring in the Internet, (submitted to ITC18).

[8] U. Hofmann, I. Miloucheva, T. Pfeiffenberger, F. Strohmeier, Evaluation of architectures for QoS analysis of applications in Internet environment, ICTSM10, Monterey, CA, USA, October 3-6, 2002.

[9] P. A. Aranda Gutiérrez, I. Miloucheva, Analysis of end-to-end Quality of Service behaviour in inter-domain environment, First international workshop on Inter-domain performance and simulation, IPS 2003, Salzburg 20-21 February 2003.

[10] Th. Pfeiffenberger, U. Hofmann, I. Miloucheva, A. Nassri, Inferencing of inter-domain path characteristics based on active end-to-end QoS monitoring – scenario studies in emulation environment, First international workshop on Inter-domain performance and simulation, IPS 2003, Salzburg 20-21 February 2003.