Maintenance Document

Linux Traffic Control-Graphical User Interface – Group paketti2

Helsinki 18th December 2004

Software Engineering Project

UNIVERSITY OF HELSINKI
Department of Computer Science

Course

581260 Software Engineering Project (6 cr)

Project Group

Fabian Fagerholm
Janne Johansson
Markku Manner
Niko Mikkilä

Client

Jukka Manner

Project Masters

Juha Taina
Marianne Korpela

Homepage

`http://www.cs.helsinki.fi/group/paketti2`

Change Log

Version	Date	Modifications
1.0	8.12.2004	First version

Contents

1 Introduction

The software development project Paketti2 was conducted as part of the course “Software Engineering Project”, held by the Department of Computer Science at the University of Helsinki. Paketti2 is the successor to the first Paketti project of 2003. The projects aimed to produce a graphical tool for easy modification of traffic control settings in the Linux kernel. The future users of the software are the various research groups at the Department of Computer Science. The university will also publish the software under either the GNU General Public License [FSF91] or the GNU Lesser General Public License [FSF99].

1.1 Purpose

The purpose of this document is to provide information and guidance for the future maintainer and developer of the software. This document serves as a guidebook for development of the software after the project.

1.2 Structure

Section 2 describes the basic architecture of the software, providing help for understanding the source code. Section 3 describes the Simple Traffic Control Configuration Protocol (STCCP) with is one of the building blocks of the software. Section 4 describes known problems regarding the software. These are issues which were found during testing, but were not corrected due to limited time. In Section 5 we cover possible future improvements to the software. This includes ideas which were presented during the development, but not implemented because of time constraints.

2 Software Architecture

Paketti is a software for managing Linux traffic control settings. The software consists of a server (pakettid) and a client (paketti). The server is installed on the systems on which the user wishes to manage traffic control settings. The serer works invoking the Linux command line traffic control tool tc, which, in turn, invokes the kernel traffic control interface. Other functions directly invoke the kernel traffic control interface.

The user interacts with the software via the Graphical User Interface, which communicates with the server using the STCCP protocol over a TCP/IP connection. An overview of the software architecture is presented in figure 1.

This section describes the basic functionality of the two software components: Client and Server.

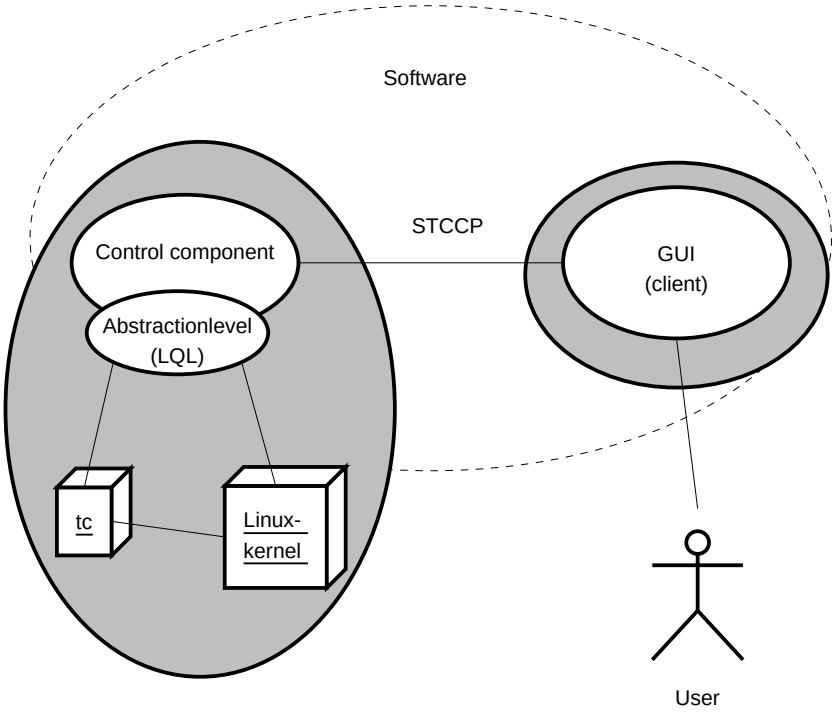

Figure 1: Overview of the software.

2.1 Client and Graphical User Interface

The Graphical User Interface, GUI, is the inferface with which the user changes the server component's traffic control settings. The GUI consists of 4 parts: a main graphical user interface part, a communications part, XML data structure and the XML and XML Schema parser part, which are all implemented using Java 1.5 [Sun04]. The main graphical user interface consists of the classes GUI, TreeNode, AttributeField, NiceBox and TextWindow. The communications part consists of the classes Connection and Message. Classes XMLAttribute, XMLAttributeType, XMLElement, XMLElementGroup and XMLElementType are used to represent the internal data structure. the XML parser part is in one class, XMLParser. Unfortunately there is no class diagram of this structure, but except for the data structure classes it is similar to the older architecture developed by the Paketti group. Figure 2 shows a representation of the old GUI architecture.

The data concerning the traffic control settings includes the following:

- A group of elements such as RED qdisc, HTB class etc.
- The values of each element's related attributes.
- The element structure, which is a tree.

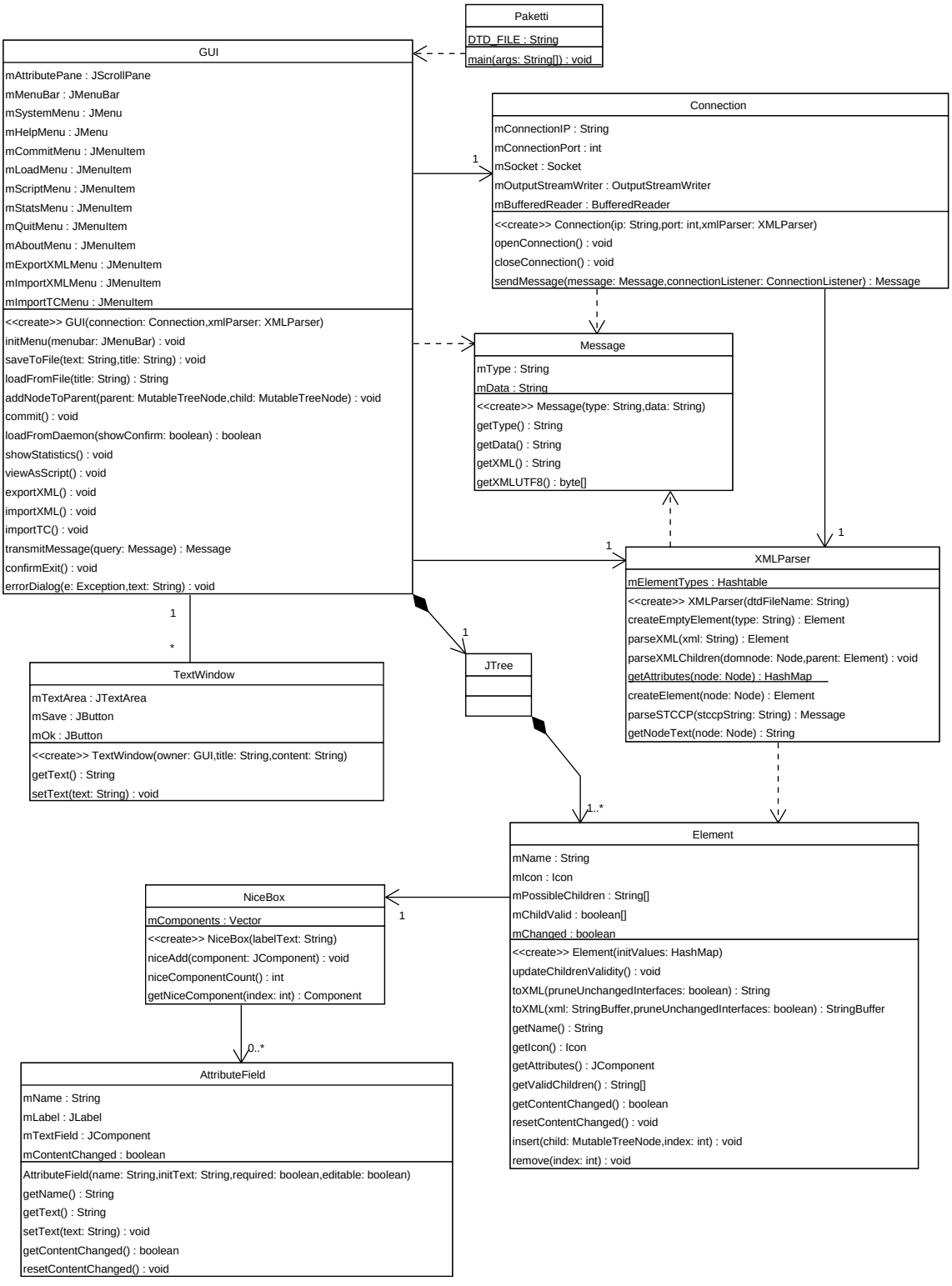

Figure 2: Old GUI architecture class diagram.

Elements are represented using the TreeNode class which is instantiated respectively to every created element. TreeNode object contains the element's related data in separate AttributeField objects. TreeNode stores the AttributeFields in a convenient graphical component NiceBox, which is the actual graphical component rendered in the GUI right hand panel. Thus every TreeNode contains a NiceBox which contains a set of AttributeField objects.

TreeNode extends Java's DefaultMutableTreeNode, which gives the ability to store the nodes in a tree structure using the DefaultMutableTreeNode method add. This tree structure is compatible with the JTree object in the GUI left hand frame and can thus be rendered directly.

AttributeField represents a single attribute or parameter value of an element visually. It has a JLabel for name and a JTextField and/or a JComboBox for data input. Additionally a help button is visible for editable attributes. AttributeField also handles unit conversions for the input data to support most input types used by the tc command.

Both TreeNode and AttributeField are backed by an internal data structure. TreeNode represents XMLElement and AttributeField represents XMLAttribute objects. The whole data structure is governed by the paketti.xsd XML Schema document. XMLElement and XMLAttribute objects contain data: XMLElement contains other XMLElement and XMLAttribute objects and XMLAttribute contains a single attribute value. The structure of these objects is however defined by separate XMLElementType and XMLAttributeType objects that represent complexType and simpleType definitions in the Schema. These definitions are parsed into XMLElementType and XMLAttributeType objects in the XMLParser class. For more details on the Schema data types, see comments in the pakettid.xsd file.

The data communication makes it necessary to encode and decode the settings data between XMLElement/XMLAttribute and XML (String) representations. An XMLElement object can return an XML representation of its subtree using the recursive method toString. Conversion to the other direction, from text to data structure is handled in class XMLParser by first using DOMParser in Xerces to parse the String into DOM nodes and then recursively creating XMLElement and XMLAttribute objects. The XML document is validated before sending it to pakettid by the Xerces XML Schema validator.

Error situations are handled using exception classes XMLParserException and ConnectionException. ConnectionException is thrown in class Connection when a communication error occurs. XMLParserException is thrown when any error occurs in class XMLParser. The exceptions are handled in GUI by invoking the method errorDialog to show the exception contained error message.

2.2 Server

The server component runs in the host computer whose traffic control settings are to be configured. It responds to the client component's requests and executes proper commands such as getting the current traffic control settings from the kernel and committing changed

traffic control settings to the kernel.

The server is started as part of the normal system init process. Upon startup, the server reads a configuration file specified by a command line parameter. Using the values obtained from the configuration file, the server locates its state file, which contains the last commited traffic control settings.

If the state file is not found, the server will assume that the previous state is the “empty” state – all traffic control settings are at their defaults. If the state file is found, the server will parse it and use the tc command and kernel interface to set up the active traffic control settings according to the state file. It will then fork a child process into the background. This child process will handle incoming STCCP connections.

After startup, the server provides the following functionality:

- Fetching the current traffic control settings and sending them back to the client in XML or TC format. (GET_SETTINGS_XML, GET_SETTINGS_TC).
- Committing traffic control settings sent by the client. (COMMIT)

The main functionality is in the file pakettid.cc, which starts the paketti daemon, pakettid. Pakettid.cc takes care of the communications with the client, using the functions in stccp.c to construct, decode, send and receive valid messages. Pakettid.cc uses the other c- and cc-files in order to serve the users requests. Pakettid can be configured in different ways, as described in the Users Manual [Pg04b]. Only one user at a time can be changing the traffic control settings of a single host.

An XSL Transformation stylesheet is used to tranform the current bandwidth settings into tc commands. The transformation is done by using libxslt. When the client executes commit, the server transforms the XML form of the traffic control settings to be committed into tc commands and executes them. The server also saves the most recent committed settings, so that it can execute those settings after the computer has been shut down. If there is an error during the commit-phase, the server component reverts to the most recent working settings. This atomic commit behaviour exists to avoid having an incomplete or non-working setup. While it protects against some errors by rolling back the settings to the previously know working state on error, it cannot prevent the user from making valid settings that have undesired effects.

3 Simple Traffic Control Configuration Protocol

Simple Traffic Control Configuration Protocol (STCCP) is an application level protocol for transmitting traffic control settings and statistics. Version 1.0 was defined by the Paketti software project group [Pg03]. Version 2.0 is defined by the Paketti2 software project group [Pg04a]. The protocol is based on the Client-Server Model architecture on TCP/IP and encodes the request and reply data in XML frames.

3.1 Model of Operation

The model of operation for transmitting the settings is as follows:

Step 1: Client opens a TCP connection to server.
Step 2: Client transmits a request to server.
Step 3: Server transmits a reply to client.
Step 4: Repeat steps 2 and 3.
Step 5: Client transmits a QUIT request and closes the TCP connection.

3.2 Message Structure

An STCCP message contains the following valid characters:

- **LF** line feed, hex decimal 0xA, decimal 10.
- **integer** a non-negative integer k in range $0 <= k < (2^{32} = 4294967296)$.
- **character** an arbitrary character of UTF-8 charset.
- **string** an arbitrary sequence of characters approved by XML syntax.

STCCP messages utilize UTF-8 charset. The messages are divided into a header and a body (XML frame). The header consists of exactly two lines, as shown:

version_number LF
length LF
xml_frame

The version_number field is of the following form:

Protocol:STCCP/integer

The value of the integer is 1 in the first version, 2 in the second and so on. The length field is of the following form:

Length:integer

The value of integer corresponds to the total length of xml_frame in bytes. The structure of the last field, xml_frame, is defined in section 3.3.

An example STCCP message of the type GET_STATISTICS_TC:

Protocol:STCCP/2 LF
Length: 54 LF
<MESSAGE><TYPE>GET_STATISTICS_TC</TYPE><DATA /></MESSAGE>

3.3 XML Frame Structure

Requests

The following defines each request along with its message type, the corresponding server reply and possible error conditions. The server replies with a message of type REPLY if the request was accepted, otherwise with type ERROR. The error conditions common to all requests are listed in the definition of ERROR reply.

HELLO

<MESSAGE><TYPE>HELLO</TYPE><DATA /></MESSAGE>

- **Description** HELLO starts the communication between the client and server. The server may ignore the message DATA element which should be empty.
- **Reply** The client may ignore the data element. Server replies either with a REPLY message, when the connection is accepted, or with an ERROR message, when the connection is rejected. The connection is rejected when there is already some user connected to the server or another error occurs.

GET_SETTINGS_XML

<MESSAGE><TYPE>GET_SETTINGS_XML</TYPE><DATA /></MESSAGE>

- **Description** GET_SETTINGS_XML requests the current traffic control settings in XML form. The server may ignore the message DATA, element which should be empty.
- **Reply** The DATA element contains the traffic control settings in XML form.

GET_SETTINGS_TC

<MESSAGE><TYPE>GET_SETTINGS_TC</TYPE><DATA /></MESSAGE>

- **Description** GET_SETTINGS_TC requests the current traffic control settings as a tc script. The server may ignore the message DATA element, which should be empty.
- **Reply** The DATA element contains the traffic control settings in a tc script. The tc commands are separated with line feeds.

GET_STATISTICS_XML

<MESSAGE><TYPE>GET_STATISTICS_XML</TYPE><DATA /></MESSAGE>

- **Description** GET_STATISTICS_XML requests statistical information in XML form. The server may ignore the message DATA element, which should be empty.
- **Reply** The DATA element contains statistical information in XML form.

GET_STATISTICS_TC

<MESSAGE><TYPE>GET_STATISTICS_TC</TYPE><DATA /></MESSAGE>

- **Description** GET_STATISTICS_TC requests statistical information in the form output by tc. The server may ignore the message DATA element, which should be empty.
- **Reply** The DATA element contains statistical information in the form output by tc.

COMMIT

<MESSAGE><TYPE>COMMIT</TYPE><DATA>string</DATA></MESSAGE>

- **Description** COMMIT requests server to run the accompanied traffic control settings into the target system. The DATA element contains the settings to be run in XML form.
- **Reply** The DATA element contains the settings that were run in XML form.
- **Error** DATA element is empty.
  DATA element contains malformed XML.
  The settings to be run contained by DATA element are invalid.
  An error occured while running the settings.

QUIT

<MESSAGE><TYPE>QUIT</TYPE><DATA /></MESSAGE>

- **Description** QUIT notifies server that the connection will be closed. The server may ignore the message DATA element, which should be empty.
- **Reply** It does not matter how the server replies. The server doesn't even have to reply.

Replies

REPLY

<MESSAGE><TYPE>REPLY</TYPE><DATA>string</DATA></MESSAGE>

- **Description** The server transmits a REPLY message in response to every accepted request except to a QUIT request. If the request type implied reply data, the data is contained in DATA element.

ERROR

<MESSAGE><TYPE>ERROR</TYPE><DATA>string</DATA></MESSAGE>

- **Description** The server transmits an ERROR message if the received request was not accepted for some reason. The reply DATA element contains additional information why the error occured.

The following possible error conditions are possible to all requets:

- The request format was invalid, and
- Server internal error occured while processing the request.

4 Known Problems

The project testing phase revealed some problems which we were unable to fix during the project. The problems are documented in this section.

4.1 Problems in Client Component

1. The GUI class has grown too large and there is quite a lot of duplicate code that could be joined together more intelligently.

2. Transform.java is hard to expand or change to handle new qdiscs and other elements. It barely manages to convert TC scripts written using the XSLT transformation. In real life TC scripts are much more complicated and importing them requires better tools. It is probably neccessary to write the whole TC->XML converter again from scratch.

4. Convenient usage of class and hashtable handles in the GUI is only partly implemented and will not work reliably. Also some field values like the TBF latency are too restricted. Others are not restricted enough. Much more intelligence should be added for the GUI to be easy to use.

5. The Help document is not finished. Also a division to separate sections for different elements is neccessary.

6. Handling of errors and warnings is not unified. Warnings are only reported in the console, which may make some tasks difficult. A status bar for showing warnings on the GUI would be most welcome. Most errors should be displayed in a dialog window, but currently there are some which are not.

4.1.1 Transform TC to XML

Transformation from tc to XML is coded in a very unpleasent way. The code is very simple, but if the syntax of the tc-script is not appropriate for the transformation-method, then there can be many incorrent values in the XML. This is partly due to the lack of documentation about tc-tool and also due to the lack of complete understanding of the tc-script syntax. If the syntax of the tc-script was accurate, then there were no errors found. See the usersmanual for specific syntax of the understood tc-syntax.

4.2 Problems in Server Component

4.2.1 Memory leaks

A substantial amount of memory leaks were detected, isolated and fixed during the testing phase. However, the software still leaks a small amount of memory. We were unable to trace the origin of these memory leaks. There are two possibilities.

- the leak is in the libxml library, or
- the leak is so severe that the stack gets badly corrupted, making it very hard to trace the real origin of the leak, but it is still in the pakettid software.

Since we were unable to fix these leaks, we will include the information obtained by a memory debugging tool (valgrind) in the hope that they might be resolved in the future.

==10426== 20 bytes in 4 blocks are definitely lost in loss record 21 of 71
==10426== at 0x1b904edd: malloc (vg_replace_malloc.c:131)
==10426== by 0x1b9a6ba5: xmlStrndup (in /usr/lib/libxml2.so.2.6.11)
==10426== by 0x1b9a6c33: xmlStrdup (in /usr/lib/libxml2.so.2.6.11)
==10426== by 0x1b9a7316: xmlStrcat (in /usr/lib/libxml2.so.2.6.11)

...

==10426== 1190 bytes in 5 blocks are definitely lost in loss record 52 of 71
==10426== at 0x1b904edd: malloc (vg_replace_malloc.c:131)
==10426== by 0x1b9a6ba5: xmlStrndup (in /usr/lib/libxml2.so.2.6.11)
==10426== by 0x1b9eac8a: xmlDocDumpFormatMemoryEnc (in /usr/lib/libxml2.so.2.6.11)
==10426== by 0x1b9eadcb: xmlDocDumpMemoryEnc (in /usr/lib/libxml2.so.2.6.11)

...

==10426== LEAK SUMMARY:
==10426== definitely lost: 1210 bytes in 9 blocks.
==10426== possibly lost: 0 bytes in 0 blocks.
==10426== still reachable: 286548 bytes in 1454 blocks.
==10426== suppressed: 200 bytes in 1 blocks.

4.2.2 The Arena memory handler

The Arena memory handler is well-intended; it is supposed to isolate all memory handling to one place. In a way, it is a primitive garbage collector that accumulates allocated memory during the execution of a program, and then frees it all after processing is done.

However, this kind of behaviour is not suitable for a long-running server process. Therefore, the Arena memory handler should be replaced. A good alternative would be the

Boehm Garbage Collector for C and C++ [Boe04], or some other garbage collector library.

Also, we have been unable to verify that the leaks mentioned in Section 4.2.1 do not originate from Arena.

4.2.3 Fundamental design flaws

The single most grave problem in the server component is its inconsistent design. The server handles internal data using several different data structures. We have been able to identify three different kinds of data structures for handling the same internal data.

The first data structure used is a memory block with null-delimited fields. This is used to pass around various kinds of data, such as tc commands, lists of interfaces, and traffic control settings in tc form. The data structure is created by asking for a fixed-size block of memory from the "arena" memory handler, implemented in arena.h and arena.c. The data block is then populated by copying char data into it.

The block functions as a primitive structure, where each member is delimited by a null character. Member access is based on either direct or dynamic access. In direct access, the fields are of a known fixed length, and access is thus only pointer arithmetic. In dynamic access, parts of the block are scanned to determine where the start of a field is. However, since the block structure is not documented, it is difficult to know when a particular block is intended to have fixed or dynamic fields.

The second data structure used is the rare occurrence of a real struct. In most parts of the code, the use of structs is very sparse, and can be easily understood.

The third kind of data structure is an XML tree provided by libxml. The XML tree represents system settings, network interfaces, TC filters, classes and queuing disciplines. In many cases, the code centers around such a tree, and consists mostly of traversing it or populating it. The use of this tree can be well understood by reading the libxml documentation and the pakettd code.

However, putting these pieces together has made the code both harder to understand as well as, in our opinion, larger. The conversion between the different data structures forms a significant part of the code. We were unable to correct this problem, since we did not understand its implications until very late in project. We have attempted to begin improving the internal architecture of the program, but did not complete this work.

It is probable that using the XML tree as the primary data structure would help this problem. Most of the operations in the pakettid server can be reduced to getting data from a source (network, kernel, file, etc.) and putting it into the XML tree, or retrieving data from the XML tree and outputting it to a destination (network, kernel, file).

4.3 Other Problems

There are several other problems that are, in one way or another, related to the design flaws described in the previous section. For example, to implement the GET_SETTINGS_XML function, the pakettid server first obtains the current TC settings by calling the tc command with the appropriate parameters. Then, the output of the tc command is parsed into the tc commands that would generate those settings. Finally, the tc commands are parsed and an XML document is created. The structure is very inflexible, and, for example, cannot actually get all settings from the tc command, since this would involve multiple invocations of the command with different parameters, which was not anticipated in the original design. Thus, pakettid cannot, for example, be easily extended to support ingress filtering. If the tc command must still be used in the future, transform.c should probably be rewritten, preferably in a higher-level language.

5 Suggestions for Future Improvements

This section makes suggestions for future upgrades on the software.

5.1 Client Component

- Improve visualization of the traffic control settings. Go from entering values into fields to directly manipulating settings by actions in the user interface. Use sliders to alter numerical settings, drag-and-drop to chain filters, classes and qdiscs, and visualize the packet processing pipeline.
- Write a web interface or a GTK+ GUI.
- Support for new languages.
- A wizard to guide beginners with the use of the client component.
- Statistics about recently made changes.
- Show traffic statictics visually.
- Ability to simulate the management settings before committing them.
- Ability to change traffic control settings on a number of hosts without having to disconnect and connect to a new host every time. Also requires improvements to the GUI.

5.2 Server Component

- Improve the internal architecture of the server.

- Remove the usage of TC-command completely and use only the linux kernel API for managing traffic control settings. Modify the server component to use for example LQL [Sie04].
- Add more filters and qdiscs.
- Add support for iptables, ifconfig and route.
- Add support for other interfaces than eth.
- Possibly split the server into a kernel library component such as LQL, and a small and simple server component written in a high-level language such as Python.

5.3 Simple Traffic Control Configuration Protocol

- Compress the STCCP-message, so it will use less space.
- Encryption of the message.
- Add authentication for the communication.
- Redefine the protocol to be line-based to avoid locking problems. The protocol could be asynchronous and use tagged commands or chunking to facilitate parallel operations.
- Alternatively, redefine the protocol to run over XML-RPC. This would automatically provide capabilities like compression, encryption and authentication, depending on the chosen implementation.

5.4 Other Improvements

- User authentication using digital signatures or passwords. However, since TC is probably always handled by users who have root access, SSH tunneling may allow sufficient authentication control. For a web interface SSL would be neccessary.

6 Vocabulary

traffic control	network traffic management to better provide QoS
client	client side of the client-server-model. See also client component.
client component	component with which it is possible to change the host computers traffic control settings. Uses a GUI.

control component	Sofware which runs in the host computer. Management settings of this computer are going to be changed.
daemon	A process which runs for a long period of time. Executes the clients requests.
Filter	tool to divide flows to proper classes.
Flow	A flow of data between client and server.
forking	a new child process which shares its parent resources.
Graphical User Interface, GUI	graphical software with which you can change traffic control settings.
Host	target computer.
Kernel	The core of the operating system.
Linux	open source operating system.
paketti	client component of the software. Also used to describe the software which includes both the paketti and pakettid.
pakettid	the server component.
Quality of service, QoS	A set of traffic control settings, which are used to improve the performance of certains flows.
queue disc, qdisc	
Reply	Reply message to the client from the server.
request	Request message from the client to the server.
server	See daemon.
server component	see pakettid.
Software	Paketti and pakettid.
STCCP	Simple Traffic Configuration Control Protocol. Protocol defined by Paketti-group and modified by Paketti2-group. Used to exchange STCCP-messages between the client and server component.
STCCP-message	A message which is defined by STCCP. Either a request or reply.
user	person who uses the client-component and possibly changes the traffic control settings of the host computer executing the control component.
XML-document	A tree-like data structure used to save and move data. Consists of inner elements, element attributes and text data.
XML-RPC	XML Remote Procedure Call. An XML- and HTTP-based communication protocol that facilitates writing distributed applications.

XML Schema	XML-language based definition language. Used to restrict XML-documents according to specified rules.
XSL	Extensible Stylesheet Language.
XSLT	XSL Transformations. Language which can be used to transform almost any XML-document to non-XML-documents, for example to PDF-documents.

References

Boe04 Boehm, H. J., A garbage collector for c and c++, 2004. URL http://www.hpl.hp.com/personal/Hans_Boehm/gc/.

FSF91 Gnu general public license, 1991. URL http://www.gnu.org/licenses/gpl.html.

FSF99 Gnu lesser general public license, 1999. URL http://www.gnu.org/licenses/lgpl.html.

Pg03 Paketti-group, Suunnitteludokumentti (in finnish), 2003. URL http://www.cs.helsinki.fi/group/paketti/dokumentit/suunnitteludokumentti.ps.

Pg04a Paketti2-group, Suunnitteludokumentti (in finnish), 2004. URL http://www.cs.helsinki.fi/group/paketti2/doc/suunnitteludokumentti.pdf.

Pg04b Paketti2-group, Users manual, 2004. URL http://www.cs.helsinki.fi/group/paketti2/doc/usersmanual.pdf.

Sie04 Siemon, D., Lql-libarary homepage, 2004. URL http://www.coverfire.com/lql/.

Sun04 Javatm 2 jdk, standard edition, 5.0, 2004. URL http://java.sun.com/j2se/1.5/docs/index.html.