

Load Balancer

Software project on the MFF UK Prague

User documentation

Thanks to:

Prof. RNDr. Milan Tichy, DrSc. for lending of ten computers in labTS laboratory of MFF UK for testing purpose.

Mgr. Jan Redl for allowing practical test in Czech OnLine company.

Contents

1	Introduction	6
1.1	Programmer's team	6
1.2	Loadbalancing problem	6
1.3	The glossary	7
1.4	Load Balancer's architecture	8
2	Installation and startup	10
2.1	Installation	10
2.1.1	Installation of ODBC interface	11
2.2	Setting of Load Balanceru	12
2.3	Setting of configuration file	12
2.3.1	Interface settings of Load Balancer	13
2.3.2	Setting of listening ports, servers and groups	15
2.3.3	Setting of first layer rules	16
2.3.4	Additional basic settings	18
2.4	Startup	18
3	Init process and Load Balancer's control	19
3.1	Load Balancer's startup	20
3.2	Load Balancer's run	22
3.3	Load Balancer's restart	22
3.4	Load Balancer's ending	23
4	Core module	24
4.1	The core architecture	24
4.2	Servers	24
4.2.1	Ports table	25
4.2.2	Table of servers	25
4.2.3	Table of groups	25
4.3	TCP and UDP threads	26
4.4	Serving of a connection	27

4.4.1	UDP connection	27
4.4.2	TCP connection	27
4.5	Choosing of server	28
4.6	Creating connection	29
4.7	Communication with statistic module	30
4.8	Ping thread	30
5	Statistic module	32
6	Configuration	34
6.1	Introduction	34
6.2	Configuration file	34
6.2.1	Basic structure	34
6.2.2	Modules, sections and aliases fundamentals	35
6.2.3	MAIN section	35
6.2.4	ALIASES section	36
6.2.5	User sections	37
6.2.6	Simple data types	37
6.2.7	Tables	38
6.2.8	Servers data type	39
6.2.9	Enumeration type	40
6.2.10	Checkings of corectness	41
6.2.11	Example of configuration file	41
7	Web interface	44
7.1	Monitoring	44
7.2	Displaying of statistic information	45
7.2.1	Creating of query	45
7.2.2	Displaying results of query	47
7.3	Configuration	48
7.3.1	Admin	49
7.3.2	The view of configuration file	49
7.3.3	Main	50
7.3.4	Aliases	50
7.3.5	Sections	51
8	Submodules	56
8.1	First layer library	56
8.2	Second layer library	58
8.3	Third layer library	60
8.3.1	Static algorithms	61

8.3.2	Dynamic algorithms	62
8.4	Data layer library	64

Chapter 1

Introduction

The Load Balancer is a school project and its goal is an implementation of a free software modular loadbalancer, which will be comparable with the most of commercial loadbalancers and in some attributes will be even better. The main focus is put on modularity, stability and performance of the program.

1.1 Programmer's team

- Lukáš Hlůže – presentation module, web interface for configuration, documentation
- Václav Nidrle – statistic module, balancing algorithms, documentation
- Přemysl Volf – Load Balancer core module, init module, balancing algorithms, documentation
- Stanislav Živný – configuration, documentation

1.2 Loadbalancing problem

The common technique of system scaling is building of servers' farms. Requests distribution among servers on farm can be done by using DNS round-robin. This method distributes requests periodically and equally among individual servers in farm. But then the request distribution among servers is not equal according to servers performance and there isn't any possibility to tune the distribution smoothly. The specific routing of certain type requests to specialized servers is impossible too.

That's why the loadbalancers are used on servers' farms. They distribute the workload among servers according to given rules, so the servers can have

a different performance, they can be optimized for different types of requests (eg. static web pages, cgi scripts, streaming servers etc.). Loadbalancer then makes a decision according to the actual configuration and different parameters like servers' actual availability, workload, performance, request type, source or destination address and port. Loadbalancer usage brings faster requests handling, requirements reduction and server failure protection.

1.3 The glossary

Module of Load Balancer is a process and so each can run on different computer. The main three modules are init, core and statistic. The core and statistic module are made up of two processes - the managing one and the executive one.

Executive process realises functions of given module.

Managing process communicates with the init process and it cares for running of executive process. It kills and restarts executive process during restart of Load Balancer. When automatic checking is turned on it checks if the executive process responds to signals. If the executive process doesn't respond the managing process restarts it.

Layer is a part of core module through which has to go incoming connections. The first, second and third layer decides to which server will be redirected incoming connections. All data go through data layer. The functions which process the decisions on individual layers are called submodules.

Submodule is a function which is saved in dll library. The submodules are called during passing of request through individual layers. The second and the third layer can call more submodules for one request.

Static algorithm is a balancing algorithm which decides according to information loaded from configuration file and information about given connection.

Dynamic algorithm is a balancing algorithm which decides according to information loaded from configuration file and it uses queries to statistic module to get appropriate statistic data.

1.4 Load Balancer's architecture

The Load Balancer is splitted into several modules. The reason is to achieve different levels of functionality and performance. Individual modules are run as separated processes to decrease possibility of Load Balancer being the bottleneck of the whole farm. The modules can run on different computers and they communicate together through TCP/IP and UDP/IP protocols.

Init process performs the Load Balancer's startup, managing and ending. Core process is the executive part that performs connection switching. Statistic process saves information about processed connections, answers questions of core process about servers' state and it provides data for presentation. The web interface provides remote Load Balancer administration and presentation of statistic information.

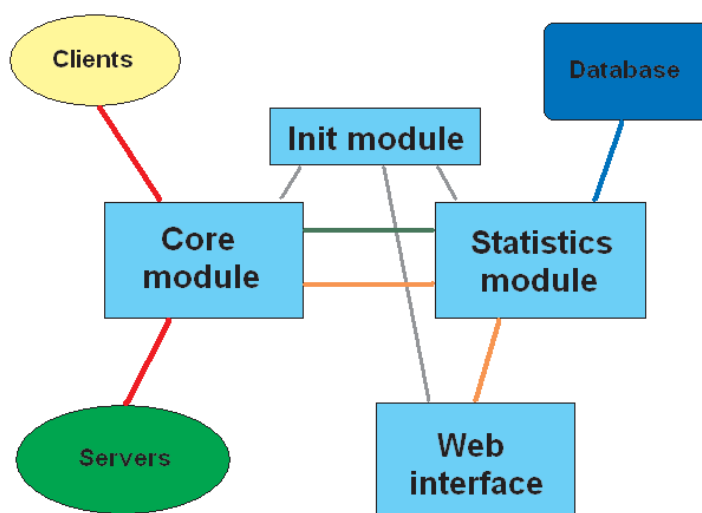


Figure 1.1: Load Balancer architecture

There is the architecture of Load Balancer and data flow between individual parts in the picture. Load Balancer is made up of light blue parts. The external database, which is connected through ODBC interface, is marked with dark blue colour. The clients, which are connecting to Load Balancer from outer network, are marked with yellow colour and the servers, which are processing requests, are marked with green colour.

The colourful joins represent the data flow. Red user data are taken to core module which redirects them to chosen server. This user data load

are very high and so these joins should be of an appropriate dimension. The grey colour represents service communication between init module and other parts of Load Balancer. It is used for sending configuration data and restarting signals. The green colour represents data sended from core to statistic module using UDP protocol. The data capacity can be regulated by settings in configuration file. The orange colour represents queries from core and web interface to statistic module. The core uses queries for balancing by dynamic algorithms, web interface uses queries for presenting statistic information about connections and servers state monitoring. The blue join represents cooperation of statistic module with external database.

The whole Load Balancer should run on local network because of its performance. The individual parts can be run in WAN, but the service communication (gray colour in the picture) is not secured and that's why it is not recommended.

Chapter 2

Installation and startup

2.1 Installation

Installation of Load Balancer from the source tarball

```
balancer-0.1.tar.bz2
```

which is located in the package directory on CD is done by the following way.

```
tar jxvf balancer-0.1.tar.bz2
```

command unpacks the source tarball. The configure and compilation scripts are run then by the sequence of commands:

```
./configure  
make  
make install
```

The last mentioned command installs the Load Balancer into the system.

The next step is the installation of php extension balancer which is needed for web interface run. The process of instalation is very similar. After unpacking the source tarball move to *present/php-ext/balancer* directory and php extension balancer is installed similarly to the Load Balancer's installation.

```
./configure  
make  
make install
```

The next step is copying of directory structure *present/www* into root directory of web server (only *webroot* from now).

The next step is copying of scripts `copy` and `restart` from the `present/bin` directory to the directory where they will be executable from the web interface, they will have rights for manipulation of `/etc/lb.cfg` and `webroot/config.cfg-default` configuration files and they will have rights for sending signal to init module. This situation can be done by the `sudo` instrument.

2.1.1 Installation of ODBC interface

If the database is used there has to be installed ODBC interface for the communication between Load Balancer and given database. Some ODBC driver manager for Unix-based system has to be installed at first (eg. `unixODBC` - <http://www.unixodbc.org>). This ODBC driver manager cares for ODBC drivers of different database servers and their connecting with Load Balancer.

Then there has to be installed ODBC driver for given database which will be used with Load Balancer (eg. MySQL Connector/ODBC driver for MySQL database - <http://dev.mysql.com/downloads/connector/odbc/3.51.html>). It processes the translation of individual requests into SQL code for given database server.

The installation can be done by usual method from the source tarballs by the sequence of commands

```
tar jxvf source.tar.bz2
./configure
make
make install
```

For successful installation is necessary to have at least this versions of libraries and tools:

glibc 2.3.4

flex 2.5.4

bison 1.875d

myodbc-3.51 3.51

unixODBC 2.2.6

php 4.3.10

gcc 3.3.5

gnu make 3.80

autoconf 2.59

2.2 Setting of Load Balanceru

It is necessary to set configuration file *webroot/config.php* for the correct run of web interface. The following variables needs to be set:

statsIP to the IP address of statistic module.

statsPORT to the port where the statictic module listens for queries from the web interface. The standard port number is 44793.

init_pid_file The name of file where the init module saves the number of its process. The web interface uses it for restarting and ending of Load Balancer. This function is not accessible if the value is empty . The standard value is */var/run/lbinit.pid*.

running_config to the name of directory where is located the configuration file readed by init module. The standard value is */etc/lb.cfg*.

bin_dir to the name of directory where are located the **copy** and **restart** scripts.

2.3 Setting of configuration file

Configuration is available using web interface now and implicit configuration file is set up. It is used always after installation of Load Balancer.

Only some basic setting needs to be changed for configuration of simple farm of servers. The individual sections are available from *Section* item in *Configuration* part.

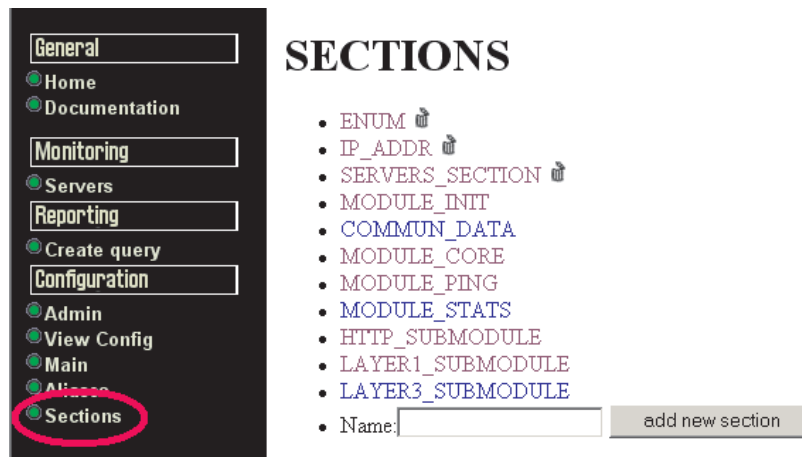


Figure 2.1: Page for setting individual sections

2.3.1 Interface settings of Load Balancer

Load Balancer's interface is set in *IP_ADDR* section. There are set IP addresses of individual parts and interface for inner and outer network in this section. IP address can be set by numbers notation (eg. 127.0.0.1) or by name (eg. localhost, www.mff.cuni.cz).

- IP_ADDR 🗑

CORE_IP (varstring) 🗑

Value:

(IP address of core module)

STATS_IP (varstring) 🗑

Value:

(IP address of statistics module)

BALANCER_INNER_IP (varstring) 🗑

Value:

(Inner IP address of Load Balancer connected to WAN)

BALANCER_OUTER_IP (varstring) 🗑

Value:

(Outer IP address of Load Balancer connected to LAN)

Figure 2.2: Interface settings of Load Balancer

The variable *CORE_IP* determines IP address of core module. The init module connects to this address.

The variable *STATS_IP* determines IP address of statistic module. The init module connects to this address.

The variable *BALANCER_INNER_IP* determines interface of Load Balancer for inner network. This address is used for translation of some protocols where it is used as a client address for servers.


The variable *BALANCER_OUTER_IP* determines interface of Load Balancer for outer network. This address is used for translation of some protocols where it is used as a server address for clients.

The core module is typically located on computer with two network interfaces. The first interface is used as IP address of core module and inner

IP address, the second interface is used as outer IP address. IP address from inner network is used for statistic module.

2.3.2 Setting of listening ports, servers and groups

Lists of ports and servers are set in *SERVERS_SECTION* section. This section includes three tables which are setting listening ports, servers and groups of servers.

LISTEN_PORTS (tabledef) 



	IP protocol	IP port
	TCP <input type="text" value=""/>	80 <input type="text" value=""/>
	TCP <input type="text" value=""/>	<input type="text" value=""/>

Figure 2.3: Setting of listening ports

Ports and protocols, where Load Balancer listens in outer network, are set in *LISTEN_PORTS* table. Clients are connecting to these ports.

SERVERS (tabledef) 



	Name	IP protocol	IP address	IP port	Static weight
	my server <input type="text" value=""/>	TCP <input type="text" value=""/>	192.168.0.3 <input type="text" value=""/>	80 <input type="text" value=""/>	100 <input type="text" value=""/>
	<input type="text" value=""/>	TCP <input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>	<input type="text" value=""/>

Figure 2.4: Setting of servers

The list of servers from farm is in the *SERVERS* table. A new record has to be created for every service. This record includes username of server, its identification (protocol, IP address and port) and static weight of server which determines a relative ratio of servers' performance. All servers should have the same static weight if their performance is the same.

GROUPS (tabledef) 


	Name	List of groups divided by comma.
	<input type="text" value="all"/>	<input type="text" value="my server"/>
	<input type="text"/>	<input type="text"/>

Figure 2.5: Setting of groups of servers

List of servers, which are used in rules of three layers, are included in the *GROUPS* table. Only one group which includes all servers needs to be set if all servers are equivalent.

2.3.3 Setting of first layer rules

The rules which are used by common function on the first layer are set in the *LAYER1_SUBMODULE* section. This section includes the *L1_RULES* table which defines individual rules.

L1_RULES (tabledef)

	Name	Send to stats	IP protocol	listen port	source port	source IP address	source mask	library on second layer	function on second layer	lib
	HTTP	YES	TCP	80	0					sta
	default TCP	YES	TCP	0	0					sta
	default UDP	YES	UDP	0	0					sta
		YES	TCP							

change

library on second layer	function on second layer	library on third layer	function on third layer	library on data layer	function on data layer	servers
		static_layer3.so	static_select_first			all
		static_layer3.so	static_select_first			all
		static_layer3.so	static_select_first			all
						all

Figure 2.6: Setting of first layer rules

There has to be set the following columns for elementary settings:

Name The name is part of identification of connection's data type in statistic module. It enables to count statistics for individual types of connections.

IP protocol It determines the protocol to which is rule related.

Listen port It determines the port where the connection was received from client (eg. 80 for HTTP protocol, 22 for SSH protocol etc.)

Library on the third layer It determines the name of library from which is function on the third layer used. The static_layer3.so library is suitable for elementary farms.

Function on the third layer It determines the name of function from the library above. The static_select_first function is suitable for elementary farms.

servers It determines the group of servers among which are connections distributed. All servers are set for elementary farms.

2.3.4 Additional basic settings

The usage of statistic module is set using the *INIT_STATS_START* variable in the *MODULE_INIT* section.

2.4 Startup

The database server should run if there is set usage of statistic module with the database. The core and statistic modules are started at first. The init module is started then and it startups the whole Load Balancer. The Load Balancer is ready for usage after initialization of all parts and it is possible to use web interface for changing configuration, managing Load Balancer and displaying statistic information about processed connections.

Chapter 3

Init process and Load Balancer's control

The whole user control of Load Balancer after it's startup is made through init process. It can be indirectly controlled through the web interface too. Settings of the init process and individual modules communication is performed in `<IP_ADDR>` section which is used by all modules and `<MODULE_INIT>` section.

List of variables in `<IP_ADDR>` section:

CORE_IP - IP address of core module.

STATS_IP - IP address of statistic module.

BALANCER_INNER_IP - Load Balancer's inner IP address which is used with some protocols switching. This address is used when the server requires address of the client.

BALANCER_OUTER_IP - Load Balancer's outer IP address which is used with some protocols switching. This address is used when the client requires address of the server.

List of variables in `<MODULE_INIT>` section (all values of *TIMEOUT* parameters are in seconds):

INIT_STATS_START - The statistic module is started when the value equals "YES", it isn't started when the value equals "NO".

INIT_CORE_PORT - The port number where the init module connects to core module, implicitly 44783.

INIT_STATS_PORT - The port number where the init module connects to statistic module, implicitly 44784.

INIT_CORE_STARTUP_TIMEOUT - The timeout value for core module startup and initialization.

INIT_STATS_STARTUP_TIMEOUT - The timeout value for statistic module startup and initialization.

INIT_INIT_CHECK_TIMEOUT - The timeout value for init control messages sending and receiving. If the value equals 0, no control is done.

INIT_CORE_CHECK_TIMEOUT - The timeout value for core control messages sending and receiving. If the value equals 0, no control is done.

INIT_STATS_CHECK_TIMEOUT - The timeout value for statistic control messages sending and receiving. If the value equals 0, no control is done.

INIT_CORE_FREEZE_CONTROL - If any value is set up, the managing process of core module performs control of core executive process run.

INIT_STATS_FREEZE_CONTROL - If any value is set up, the managing process of statistic module performs control of statistic executive process run.

INIT_SYSLOG_LEVEL - Setting of system logging level for all modules. The implicit value is *LOG_ERR*.

3.1 Load Balancer's startup

At first there are started up the core and statistic modules (if the usage of statistic module is set up). Both processes can be run with the following options:

-d Don't run process as a daemon. All processes are implicitly run on background as daemons.

-p PORT Core (statistic) process will listen for connection accepting on the PORT port. Implicitly it listens on 44783 (44784) port.

-l IP_ADDR Core (statistic) process will listen for connection accepting from *IP_ADDR* address. Implicitly it listens on *INADDR_ANY* address.

The module waits on init connection after startup and options processing. The init module can be run with the following options:

- d** Don't run process as a daemon. Init process is implicitly run on background as a daemon.
- f FILE** Use *FILE* file as configuration file. Implicitly it uses */etc/lb.cfg* file.
- o** Make only the control of configuration file and don't startup Load Balancer.
- a** All errors are found in configuration file. Implicitly the configuration file's checking stops after the first found error.
- e** Write found errors to stderr output. Errors are implicitly written only into the system log.

Init module after startup handles its options, reads and controls configuration file and establishes communication with the core and statistic modules. It sends them the configuration file and waits on completion of their initialization. During the startup sequence are both core and statistic module divided into two processes. Parent process is the managing one, it provides communication with init module and eventual checking of child process run.

When the configuration file is corrupted, init module reports the position of found error and ends up. It is possible to send found errors to stderr output and terminal by setting the appropriate options. It is processed only the first syntax error (eg. missing semicolon, missing quotes, wrong number of parameters in *ALIASES* section etc.), because parser can't continue working. In that case is written the number of row where an error was found. User can find this error on given row or one of preceding rows. This situation is explained by following example:

```
30: varint "MAX" "100" "" ""
31:
32: # backing up
33:
34: varbool "backup" "NO" "" "";
```

The semicolon is missing on row number 30. The parser skips over rows number 31 and 33 (because they are empty) and row number 32 (because it is a comment). There is found a keyword (varbool) on row number 34 instead of the semicolon. That's why the error is reported to be on row number 34. So the user checks the row number 34 which is OK. Then he checks rows towards configuration file beginning, ignores empty rows and comments and finally finds error on row number 30.

The missing semicolon in *MAIN* or *ALIASES* section is reported as error of bad or nonexisting assignment.

```
<MAIN>
core coremain coreext2
stat statisticssection;
</MAIN>
```

In this case it is missing semicolon on the first row of *MAIN* section and the next word which is read by the parser is *stat*. The parser considers this word for next user section which is assigned to *core* module. An error is reported if no such section exists. The same behaviour has the missing semicolon in *ALIASES* section.

The second part of configuration file checking is eg. the proper number of parameters in user sections, existence of sections which take place in aliases definitions etc. It is possible to set up if only the first error or all errors are reported for this part of checking.

3.2 Load Balancer's run

The Init module checks run of the core and statistic module if the appropriate parameters are set. If the contact with modules is lost, it waits on their restart or connection restoration and according to Load Balancer's state sets all it's parts to normal state. The managing processes are checking executive processes if this control is set. If the executive process doesn't respond to managing one, it is treated as being non-functional. Managing process ends run of executive process, startups it again and does it's initialization. No signals should be sent to the processes except the init during it's run.

3.3 Load Balancer's restart

When the configuration file is changed, the whole Load Balancer has to be restarted. It is done by SIGHUP signal sending to init process or by clicking to the appropriate button in the web interface. The core and statistic

processes are ended and started up again with new configuration file during restart. The Load Balancer isn't working during restart so it occurs failures of current connections and no new connections are accepted during restart.

3.4 Load Balancer's ending

The Load Balancer is ended by sending the SIGTERM signal to the init process or by clicking to the appropriate button in the web interface. Init process sends the message for modules ending to the core and statistic modules and it ends up itself. Managing processes ends up executive processes and ends up themselves after receiving the message for modules ending.

Chapter 4

Core module

The core module is the executive part of Load Balancer which manages connections switching. It transfers the incoming connection to the appropriate UDP thread or it chooses one of free TCP threads which serves the connection.

4.1 The core architecture

The main focus of Load Balancer is given to the modularity, scalability, stability and performance. It is designed to execute at least the basic operations when some other parts of Load Balancer aren't running or after their failure.

The main focus during the connection switching is given to modularity and performance. The libraries in three layers are used for choosing the appropriate server from the farm. They can be added and loaded during run of the Load Balancer. Using of the new library needs changes in configuration file and so restart of the Load Balancer too. The core module performance can be set by the automatic managing of serving TCP threads count according to configuration file settings. So there is a possibility of compromise between the performance and amount of used system resources according to the actual Load Balancer's workload.

4.2 Servers

The table of servers takes part in `<SERVERS_SECTION>` section of configuration file. This section includes `LISTEN_PORTS`, `SERVERS` and `GROUPS` tables. The names of servers and their groups occur like the listbox wherever the type varservers is used.

4.2.1 Ports table

The table *LISTEN_PORTS* includes the list of ports which are opened for listening by the core module. Clients from outer network are connecting to those ports. The only exception are the ports opened for helping data connections in data layer submodule. The table includes two columns:

protocol The string with values TCP or UDP. It specifies the protocol of incoming connection.

port The number specifies the opening port.

4.2.2 Table of servers

The table *SERVERS* includes list of servers among which are the incoming connections switched. Every individual service on the server has to be in the independent record of table. The table includes five columns:

name The name of server which will be shown in the list of names in variables of varservers type.

protocol The service protocol, the value of string is TCP or UDP

IP address The string including IP address of given server.

port The number specifying the server's (service's) port.

static weight The number specifying the static weight of service. The ratio of connections assigned to the server is equal to the static weights among servers of the group chosen for specific connection (if the appropriate algorithm is chosen).

4.2.3 Table of groups

The table *GROUPS* includes the list of groups. Every group includes the list of servers from the *SERVERS* table. The individual server can be included in several groups. The table *GROUPS* includes two columns:

name The name of group which will be shown in list of names in variables of varservers type.

list of servers Data type varlist which includes list of server's names from the table *SERVERS*.

4.3 TCP and UDP threads

The core module works in more threads simultaneously to increase its performance. A new thread is created for every opened port of UDP protocol. The number of TCP threads is managed by the appropriate variables settings in the configuration file. The number of TCP threads is being increased and decreased during run of Load Balancer according to the actual workload of core module.

The number of TCP threads equals to *CORE_THREADS_INIT* value after the startup of Load Balancer. It can increase to *CORE_THREADS_MAX* value during sufficient Load Balancer workload and it can decrease to *CORE_THREADS_MIN* value during low workload. When the *CORE_THREADS_MIN* value is lower than *CORE_THREADS_INIT* value the threads count would decrease to *CORE_THREADS_MIN* value immediately after Load Balancer's startup because the traffic wouldn't reach its standard value yet. That's why the first decrease of threads number can take part after the first increase of this value.

The core module checks the ratio of existing threads number to actually working threads number after accepting new connection. The core module tries to increase threads count when this ratio gets over the *CORE_THREADS_FULL* value. It multiplies the actual threads count by *CORE_THREADS_RATE_UP* coefficient which should be greater than 1. When the new threads count is greater than *CORE_THREADS_MAX* it is decreased to this value. The core module tries to decrease threads count when this ratio gets lower than *CORE_THREADS_EMPTY* value. It divides the actual threads count by *CORE_THREADS_RATE_DOWN* coefficient which should be greater than 1. $1/CORE_THREADS_RATE_DOWN$ should be greater or equal to *CORE_THREADS_EMPTY* at the same time. When the new threads count is lower than *CORE_THREADS_MIN* it is increased to this value.

The variables *CORE_THREADS_FULL* and *CORE_THREADS_EMPTY* should be from the $\langle 0, 1 \rangle$ interval.

The list of all variables that are influencing the TCP threads count:

- CORE_THREADS_INIT*** The initial value of TCP threads count after startup of Load Balancer.
- CORE_THREADS_MIN*** The minimal value of TCP threads count.
- CORE_THREADS_MAX*** The maximal value of TCP threads count.
- CORE_THREADS_FULL*** The ratio of existing and actually working

threads count when the core module tries to increase TCP threads count.

CORE_THREADS_RATE_UP The TCP threads count is multiplied by this value during increasing of their count.

CORE_THREADS_EMPTY The ratio of existing and actually working threads count when the core module tries to decrease TCP threads count.

CORE_THREADS_RATE_DOWN The TCP threads count is divided by this value during decreasing of their count.

4.4 Serving of a connection

4.4.1 UDP connection

It can be differentiated the beginning, but it can't be differentiated the end of communication using UDP protocol, because the communication is not binded and proceeds by individual packets. That's why it is necessary to let communication socket opened for possible next communication. It brings the problem of open sockets count continual progress.

This problem is solved by array with data about opened sockets which is *CORE_UDP_CONNECTIONS* long. The oldest connection (with the oldest date of last packet) is closed and a new socket is opened for new connection when given array is full.

The appropriate server is chosen through the use of algorithms on three layers after receiving of the connection. There is opened socket to the chosen server and received data are redirected there.

4.4.2 TCP connection

The core module waits on new requests in the main thread. The function listen is called on all opened ports and after occurrence of new connection is woken up one TCP thread for serving the connection.

The minimum of basic thread operations is done after waking up the TCP thread to process next connection as fast as possible.

The appropriate server is chosen through the use of algorithms on three layers after receiving of the connection. The connection is closed and thread is prepared to receive next request, if no server is chosen or if all servers are down.

4.5 Choosing of server

The choosing of server proceeds the same way for TCP and UDP protocols. The thread calls submodule functions on all three layers. Their output is the list of servers with priorities according to which is decided the sequence of servers for connection switching.

The function of submodule on the first layer is called for every connection and it is compulsory. The *CORE_LAYER1_FUNCTION* variable includes the name of function and the *CORE_LAYER1_LIBRARY* variable includes the name of library where given function is saved. The function can use a lot of parameters such as port number, protocol, IP address of client etc. The output is the group of servers, which can serve given request, library and function on second or third layer or data submodule and the readed data.

The function of submodule on second layer is called according to the output of the first layer function. There is typically special submodule for every port/protocol. This layer clarifies the group of servers which can serve given request and which were taken from the first layer. The output is the library and function for the third layer, possible data submodule and the readed data. The second layer submodules decides typically according to readed data (Layer 7 switching). The functions of second layer modules can be called repeatedly for one request and so smooth the dividing. There is implemented the submodule for balancing according to data content for HTTP protocol.

After going through the first two layers should be chosen group of servers equivalent in capability of processing the request of specific connection type. There is determined the sequence of servers according to their static parameters (static weight, source IP address, ...) or using queries to statistic module in the third layer submodule functions. The core tries to progressively create connection to given servers according to thier priority values. The third layer submodule function shouldn't be called only if the given request can be served only by one specific server or the servers' priority is set from previous layers.

The thread starts to choose the destination server according to list of servers after going through all three layers. The list of servers includes priority value which resolves the sequence of servers. The first server from the list with the highest priority value is chosen and the thread tries to connect to it. If the connection is not successful the number of unsuccessful connections is increased for the server and the thread continues to find another server with the highest priority. If there is no such server it chooses the server with the second highest priority value etc. until the connection is successful or the highest priority value equals 0. The server is marked to be inaccessible if the

number of unsuccessful connections exceeds the *PING_TDD* value.

The data submodule can be used only for TCP connections for managing of new connections and modification of transferred data. The data module is loaded and used if one of three layer submodules sets it. It is called during the creation or finishing of the main connection and during data transfer. The data are readed, the data submodule function is called and then the data are written. The data submodule can change data before writing them and it can create, open and close helping data connections from and to server. All data connections are finished after finishing the main connection. There is implemented the data submodule for FTP protocol in the Load Balancer which allows active and passive modes.

All submodules are loaded as dll libraries and it is possible to add support for new protocols by changing the existing or creating new dll libraries. Load Balancer needs to be restarted for accepting changes. This approach provides high modularity of Load Balancer and possibility of adding extensions for special protocols.

4.6 Creating connection

Creating of a connection is done by calling *connect()* function which has too long timeout set in the system and this could block the whole Load Balancer. That's why there are several settings which can shorten implicit timeout. There are used following variables for settings:

CORE_CONNECT_TIMEOUT It determines the length of timeout in milliseconds. If this value is 0, the autoconfiguration is used.

CORE_CONNECT_SEND_TO_STATS Information about every connect are sended to statistic module if this variable is true.

CORE_CONNECT_STARTING_TIME It determines the starting length of timeout for autoconfiguration.

CORE_CONNECT_NUMBER It determines the number of connections representing the average value.

CORE_CONNECT_RATE It determines the value which should multiply the average value for length of timeout setting.

The core module counts average value of time of connection to server when autoconfiguration is used. The average value is counted from *CORE_CONNECT_NUMBER* values where actual connection time

and *CORE_CONNECT_NUMBER - 1* values of previous average are used. The *CORE_CONNECT_STARTING_TIME* value in milliseconds is used as starting average value. After the setting of timeout of connection is counted average value multiplied by *CORE_CONNECT_RATE* value. The resultant formula:
$$\text{new_average_connect_time} = \frac{\text{average_connect_time} * (\text{CORE_CONNECT_NUMBER} - 1) + \text{new_connect_time}}{\text{CORE_CONNECT_NUMBER}}$$
 where *average_connect_time* is an average time of previous connects and *new_connect_time* is time of connect for actual connection.

4.7 Communication with statistic module

The information data about the connection is sended to statistic module if this module runs and there is set up the appropriate parameter for sending this data in configuration file. This data are used for answering queries on actual servers workload from submodules and for counting statistics for different time intervals and their presentation on web interface.

One piece of information is sended after redirecting packet to server and one after redirecting packet to client for UDP connections. The information are sended in the following cases for TCP connections:

- Creating of new connection.
- Data transfer from client to server.
- Data transfer from server to client.
- Ending the connection by client.
- Ending the connection by server.
- Data transfer from client to server through the helping data connection.
- Data transfer from server to client through the helping data connection.

UDP connection on *COMMUN_STATS_PORT* port is used for sending these information to statistic module.

4.8 Ping thread

The ping thread is created if there is set *PING_ENABLE* variable in configuration file. It goes through the list of servers and makes accessibility test on

given port for every one. The message is sended to statistic module if there is a change of server's state. The list is done every *PING_DELAY* miliseconds.

The Load Balancer includes testing for the following services: SMTP, HTTP, MySQL, FTP, POP and IMAP. Tests of other services can be added in ping.c library which is described in details in the programmers documentation.

Chapter 5

Statistic module

The statistic module is used for saving data and information about Load Balancer processing. These information are available to the user through the queries from web interface and the core module of Load Balancer can use them for balancing of incoming requests. Data are saved into database typically which communicates with the Load Balancer using ODBC interface and so it is possible to use different database servers. There are saved aggregated data in time intervals of different dimensions(second, minute, hour, day, month, year) in database, there is set different time of data duration for every time dimension in configuration file not to store too old and detailed data. Data can be duplicated and saved into memory buffer too which is used for fastening queries from the core module on recent data. The usage of database is not compulsory and data can be saved only in memory.

The individual parameters of statistic module can be set in configuration file in the `<MODULE_STATS>` section. The parameters are following:

STATS_QUERY_MODE It determines if the data for processing answers to queries from core module are located in database and database buffer (the value of parameter is "database") or if there is a memory buffer for processing these queries (the value of parameter is "memory").

STATS_USE_DATABASE It determines if data are saved into database or only into memory.

STATS_DB_NAME The name of database where data are saved.

STATS_DB_USER The Load Balancer connects to database using this username.

STATS_DB_PASSWORD The database password for user *STATS_DB_USER*.

STATS_DB_BUFFER_INTERVAL_COUNT The number of seconds for which are data aggregated and saved in memory buffer before they are moved into database (if the database is used) or before they are erased from memory buffer (if the database is not used). The recommended values of parameters are very low when database is used (2-3). When database is not used it is set according to how old data user wants to show.

STATS_MIN_SECOND_DATA_DURATION Minimal time duration of data aggregated in second intervals before they are erased from database. The value is in minutes.

STATS_MIN_MINUTE_DATA_DURATION Minimal time duration of data aggregated in minute intervals before they are erased from database. The value is in hours.

STATS_MIN_HOUR_DATA_DURATION Minimal time duration of data aggregated in hour intervals before they are erased from database. The value is in days.

STATS_MIN_DAY_DATA_DURATION Minimal time duration of data aggregated in day intervals before they are erased from database. The value is in months.

STATS_MIN_MONTH_DATA_DURATION Minimal time duration of data aggregated in month intervals before they are erased from database. The value is in years.

STATS_BUFFER_SIZE Size of memory buffer used for processing answers to queries from core module if this buffer is used (*STATS_QUERY_MODE* = "memory"). The size is in Bytes and is divided among all servers.

Chapter 6

Configuration

6.1 Introduction

The individual modules can run on different computers, but the configuration file is located only on one computer where the init module is running. The standard name of configuration file is *lb.cfg* and its standard location is in */etc* directory. These standard values can be changed during instalation or by setting of appropriate init module's options (see chapter Init process and Load Balancer's control).

The individual modules of Load Balancer are transferring configuration file content to each other. Load Balancer's configuration can be done by direct changes in configuration file and restart of init module or by using web interface.

6.2 Configuration file

6.2.1 Basic structure

The configuration file consists of compulsory section MAIN, compulsory section ALIASES and optional user sections (in given order). The beginning of the SEC section is introduced by

```
<SEC>
```

```
    row and finished by
```

```
</SEC>
```

row, where SEC can be substituted by MAIN, ALIASES or any identifier of user section. Allowed identifier of user section is any sequence of tokens

which starts with underline sign or with english alphabet character and continues with any english alphabet character, digit or underline sign. Every row in any section is ended by semicolon and configuration file is case-sensitive. Commenting rows starts with # sign, but these rows are removed from configuration file after changes made using web interface. The special mechanism was designed for commenting individual variables from configuration file, this mechanism will be described below.

The overall concept of configuration file was designed to easily extend Load Balancer with new modules which can obtain appropriate data from configuration file in simple way.

6.2.2 Modules, sections and aliases fundamentals

The configuration file is used for saving modules' parameters. User sections are used for naming of a specific area where are some parameters saved. They were created for saving several alternative settings of internal parameters and simple switching among them.

The MAIN section enables assigning user sections to modules. If given module requests data from configuration file and identifies itself by name, it receives values of variables from all user sections which are set to module of given name.

User sections can be shared among modules, in general there can be one user section assigned to more modules.

Alias concept was created for better user convenience. Alias can be taken as "several rows in MAIN section". It is an abbreviation for in general more assignments where each of them assigns user section to module. The main reason for creating of aliases was the possibility of quick and easy switching among different balancing algorithms (see example of configuration file below), the change of appropriate alias in MAIN section is enough for it.

The section ALIASES is used for definition of aliases which can be used in the MAIN section.

6.2.3 MAIN section

The MAIN section is used for assignment of user sections to given module as it was already said above. This can be done directly or by using aliases. There are two types of rows in MAIN section. The first type are rows including single word and semicolon which says what alias should be used. The resultant effect is the same as if all rows from ALIASES section which are beginning with the given alias were inserted into the MAIN section. The second type are more than one words in the section MAIN row. Then it is

interpreted as assignment of user sections (all words except of the first one) to the module (first word).

The MAIN section is compulsory, but it can be empty too. Without this section is configuration file claimed as syntactic wrong.

If there are assigned identifiers of user sections to the same module on more than one row, the resultant assignment is made up of union of given user sections. The same rule is applied to the MAIN section.

The MAIN section example:

```
<MAIN>
  core core_base core_ext SERVERS_SECTION;
  stat statistics SERVERS_SECTION;
  round_robin1;
  prezent2;
</MAIN>
```

6.2.4 ALIASES section

The ALIASES section is compulsory, but it can be empty in the same way as the MAIN section. Without ALIASES section is configuration file claimed as syntactic wrong.

The ALIASES section is used for definitions of aliases which can be used in the MAIN section.

Every row from ALIASES section has to include at least three words. The first one is the name of alias, the seconds stands for the name of module and other words are the names of user sections which are assigned to given module.

There are ignored all aliases which are included in MAIN section and are not defined in ALIASES section.

The ALIASES section example:

```
<ALIASES>
  round_robin1 mod1basic mod1A mod1C;
  round_robin1 mod2php PHP;
  round_robin2 mod1basic mod1B mod1C;
  round_robin2 mod2php PHPnew;
  prezent1 prezent prezent1 prezent_old_style;
  prezent2 preeznt prezent1 prezent_new_style;
</ALIASES>
```

6.2.5 User sections

The user section is label of section which includes several variables. They are used for saving several alternative configurations which can be switched easily or for sharing sections among modules. The examples of user sections are shown below in the complet configuration file example.

The configuration file is checked for correct syntax (regular count of arguments, non-empty identifiers etc.), but when there are found two sections with the same name they are concatenated into the single section. But if their content isn't disjunct ,eg. two definitions of the same variable, the module receives the value from the first definition and in general the behaviour isn't definite.

All rows of user section have the same structure: the first is keyword and then several values in quotation marks ended up with the compulsory semicolon. The count of values after keyword depends on the given keyword.

6.2.6 Simple data types

There is presented the list of simple data types below together with according keyword for declaration of variable of given data type and implementation data types in C programming language.

There are supported these data types:

- **varint** - integer (*int*)
- **varlint** - long integer (*long*)
- **vardouble** - floating point numer (*double*)
- **varchar** - character (*char*)
- **varbool** - truth value (*int*)
- **varstring** - string (** char*)
- **varlist** - list (**char[]*)

All these types have the same style of notation in configuration file, they differ only in the keyword. The given row has to be in form

```
keyword "identifier" "value" "range" "comment";
```

or

```
keyword "identifier" "value" "range" "comment" "longcomment";
```

so it includes one of the keywords listed above and four or five values in quotation marks ended up by the semicolon.

The meaning of individual values:

- *identifier* - the identifier of variable, non-empty string
- *value* - the value of variable, can be empty string (evaluation see below)
- *range* - the range of values (list of values separated by commas, enumeration type see below)
- *comment* - text description of variable for web interface
- *longcomment* - detailed text description of variable for web interface (optional)

It is shown here how to comment variables in configuration file.

Evaluation of variables' values. If values of **varint**, **varlint** and **vardouble** data types doesn't represent number, the resultant value is 0. The data type **varchar** returns the first character of given string. The data type **varlist** is a string of values separated by commas. The data type **varbool** can obtain value 1 (true) if the string equals "YES" or "TRUE" (values are not case-sensitive) or given string is evaluated by *atoi()* function to return positive value.

The example of section which contains definitions of all simple data types:

```
<prezent1>
varint    "MAX" "100" "" "max threads" "";
varlint   "EE" "20000" "" "";
vardouble "pi" "3.14" "" "";
varchar   "zn" "a" "" "";
varbool   "print_all" "yes" "" "print all details";
varstring "email" "ab@cd.net" "mail" "in case";
varlist   "days" "mo,tu" "mo,tu,we,th,fr" "service" "backup days";
</prezent1>
```

6.2.7 Tables

There are the following keywords for definition of tables:

- **tabledef** "id" "n" " t_1 " ... " t_n " - defines table of given name (id), number of columns (n) and data types of individual columns (t_1 , ..., t_n). There can be used only simple data types.

- **tablerow** "id" "v₁" ... "v_n" - the row of table with individual values (v₁, ..., v_n)
- **tablehelp** "id" "comment" - description of table for web interface
- **tablecomm** "id" "c₁" ... "c_n" - description of individual columns of table for web interface
- **tablelcomm** "id" "c₁" ... "c_n" - detailed description of individual columns of table for web interface
- **tabledlist** "id" "l₁" ... "l_n" - the range of values for columns of a table (used for droplist in web interface). It is similar to *range* of simple data types, it is a list of values separated by commas (enumeration types see below).

There is the compulsory declaration of table using **tabledef** and insert of data using **tablerow**. Other parts are optional, but it should be used because of web interface. All definitions of one table should be included in one user section.

The example of section using data type table:

```
<prezent2>
tabledef "rules" "3" "varint" "varlist" "varstring";
tablecomm "rules" "priority" "list of days" "email";
tablelcomm "rules" "" "" "";
tabledlist "rules" "1,2,3,4,5" "mo,tu,we,th,fr" "";
tablehelp "rules" "rules where error messages are sended";
tablerow "rules" "2" "mo,we" "ab@cz.net";
tablerow "rules" "1" "mo,tu,fr" "root@localhost";
</prezent2>
```

6.2.8 Servers data type

The main part of configuration file for the most of balancing modules is the table with rules how and where should they redirect the incoming requests. One of the columns in this table includes the server(s). There was created special data type with keyword **varservers** for user to make his work with the web interface more convenient. The variables of this type are declared in the same way as variables of simple data types

```
varservers "identifier" "value" "range" "comment";
```

or

```
varservers "identifier" "value" "range" "comment" "longcomment";
```

so the fifth argument (detailed description) is optional. Other arguments have the same meaning as those with simple data types.

So why is this data type special? The only allowed values are the name of server or the name of groups of servers (for using of more servers it is necessary to create group of servers). The value of *range* parametr is wrong here.

The data type **varservers** can be used for simple type definition or for column of table.

The *SERVERS* section example:

```
<SERVERS_SECTION>
  tabledef "LISTEN_PORTS" "2" "varstring" "varint";
  tablerow "LISTEN_PORTS" "TCP" "11230";
  tablerow "LISTEN_PORTS" "TCP" "11231";

  tabledef "SERVERS" "5" "varstring" "varstring" "varstring" "varint"
"varint";
  tablerow "SERVERS" "serv0" "TCP" "195.113.27.248" "1111" "100";
  tablerow "SERVERS" "serv1" "TCP" "195.113.27.248" "1112" "200";

  tabledef "GROUPS" "2" "varstring" "varstring";
  tablerow "GROUPS" "all" "serv0,serv1,serv2,serv3,serv4,serv5";
  tablerow "GROUPS" "group1" "serv0,serv1,serv2";
</SERVERS_SECTION>
```

6.2.9 Enumeration type

The enumeration type is defined by using keyword **defenum**, which expects type identifier and list of values separated by commas. The example:

```
defenum "work_days" "mo,tu,we,th,fr";
defenum "czech_days" "po,ut,st,ct,pa,so,ne";
```

Defined enumeration types can be used in *range* for given variable or column in table. If some value in *range* starts with ‡ sign it is considered to be the enumeration type. If no such enumeration type definition is found in configuration file, it is treated as normal string value.

If it is used enumeration type in configuration file, its definition is being seeked in a special ENUM section and than in all user sections in given

configuration file. All enumeration types should be included in user section ENUM for faster processing.

The example of enumeration type usage:

```
<core_base>
  varstring "day" "mo" "#work_days,sa,su";
</core_base>

<ENUM>
  defenum "work_days" "mo,tu,we,th,fr";
  defenum "czech_days" "po,ut,st,ct,pa,so,ne";
</ENUM>
```

6.2.10 Checkings of corectness

The Load Balancer checks the configuration file before its startup. It checks syntactic corectness, the right number of arguments after given keyword, the right number of columns in given table, known data types etc. It doesn't check semantic corectness, the user has the full responsibility.

The starting Init modul can be run only to check corectness of configuration file (see chapter Init process and Load Balancer's control).

6.2.11 Example of configuration file

Its the complex example of the whole configuration file.

```
<MAIN>
  core core_base core_ext SERVERS_SECTION;
  stat statistics SERVERS_SECTION;
  round_robin1;
  prezent2;
</MAIN>

<ALIASES>
  round_robin1 mod1basic mod1A mod1C;
  round_robin1 mod2php PHP;
  round_robin2 mod1basic mod1B mod1C;
  round_robin2 mod2php PHPnew;
  prezent1 prezent prezent1 prezent_old_style;
  prezent2 preeznt prezent1 prezent_new_style;
</ALIASES>

<core_base>
```

```

    varstring "day" "mo" "#work_days,sa,su" "backup day";
</core_base>

<core_ext>
    varint "core_n" "2" "1,2" "number of copies";
</core_ext>

<mod1A>
    varint "A" "10" "" "";
</mod1A>

<mod1B>
    varint "A" "20" "" "";
    varint "size" "10" "" "";
</mod1B>

<mod1C>
    varint "C" "100" "" "";
</mod1C>

<mod1D>
    varint "C" "200" "" "";
    varint "size" "5" "" "";
</mod1D>

<PHP>
    varstring "php" "3.0" "" "";
</PHP>

<PHPnew>
    varstring "php" "4.0" "" "";
    varstring "php2" "3.0" "" "";
</PHPnew>

<prezent_old_style>
    varlist "colors" "black,white" "" "";
</prezent_old_style>

<prezent_new_style>
    varlist "colors" "red,blue,white" "" "";
</prezent_new_style>

<prezent1>

```

```

varint    "MAX" "100" "" "max of threads" "";
varlint   "EE" "20000" "" "";
vardouble "pi" "3.14" "" "";
varchar   "zn" "a" "" "";
varbool   "print_all" "yes" "" "print all details";
varstring "email" "ab@cd.net" "email to admin" "use in emergency";
varlist   "days" "mo" "mo,tu,we,th,fr" "service days" "backup";
</prezent1>

<prezent2>
tabledef  "pravidla" "3" "varint" "varlist" "varstring";
tablecomm "pravidla" "priority" "list of days" "email";
tablelcomm "pravidla" "" "" "";
tabledlist "pravidla" "1,2,3,4,5" "mo,tu,we,th,fr" "";
tablehelp "pravidla" "addresses where error messages are sent";
tablerow  "pravidla" "2" "mo,we" "ab@cz.net";
tablerow  "pravidla" "1" "mo,tu,fr" "root@localhost";
</prezent2>

<SERVERS_SECTION>
tabledef "LISTEN_PORTS" "2" "varstring" "varint";
tablerow "LISTEN_PORTS" "TCP" "11230";
tablerow "LISTEN_PORTS" "TCP" "11231";

tabledef "SERVERS" "5" "varstring" "varstring" "varstring" "varint"
"varint";
tablerow "SERVERS" "serv0" "TCP" "195.113.27.248" "1111" "100";
tablerow "SERVERS" "serv1" "TCP" "195.113.27.248" "1112" "200";

tabledef "GROUPS" "2" "varstring" "varstring";
tablerow "GROUPS" "all" "serv0,serv1,serv2,serv3,serv4,serv5";
tablerow "GROUPS" "group1" "serv0,serv1,serv2";
</SERVERS_SECTION>

<statistics>
varint "max" "15" "" "";
</statistics>

<ENUM>
defenum "work_days" "mo,tu,we,th,fr";
defenum "czech_days" "po,ut,st,ct,pa,so,ne";
</ENUM>

```

Chapter 7

Web interface

7.1 Monitoring

There is possible to display the actual state of all servers (services) of farm in the monitoring part of web interface. The displayed information are the following:

name The name of server which is assigned in the configuration file.

IP address IP address of given server.

port Port where the service is running.

state The actual state of given server.

This is an example of servers' state display:

Name	IP address	Port	State
test 1	195.113.17.191	80	✓
test 2	195.113.21.150	80	✗
test 3	195.113.20.12	80	?

Figure 7.1: State of servers

The server *test 1* is correctly running, state of server *test 3* is unknown (eg. the last connect to server timeouted) and there occurred some failure on the server *test 2* and it doesn't work.

It is possible to show all servers which were used on the farm and needn't be used now (so they are not set in the configuration file). These servers can be displayed only if database is used for saving statistic information.

The actual state of servers in the monitoring part is refreshed every 10 seconds. No data are displayed when the statistic module is not accessible.

7.2 Displaying of statistic information

There is possible to display different statistic information about work of Load Balancer in this part. User creates some query where he specifies what data and in what time interval he wants to display. The result is displayed in the graph and in the table of individual values after sending the query. The resultant data are refreshed every minute.

7.2.1 Creating of query

In the first part the user can choose from the list of servers and their data types about which of them he wants to display information. There has to be chosen at least one data type of some server to create query.

service :

Name	IP address	Port	data_type	
serv0	212.20.96.20	80	def_tcp	<input type="checkbox"/>
			tcp	<input checked="" type="checkbox"/>
			www	<input type="checkbox"/>
serv1	212.20.96.22	80	www	<input type="checkbox"/>
			tcp	<input checked="" type="checkbox"/>
serv2	212.20.96.23	80	www	<input type="checkbox"/>
			tcp	<input checked="" type="checkbox"/>
serv3	212.20.96.24	80	www	<input type="checkbox"/>
			tcp	<input checked="" type="checkbox"/>

Figure 7.2: Choosing of servers and data types

The preceding example shows choosing data type *tcp* of all servers for displaying information.

Note: The names of data types are created from names of protocols and balancing rules from configuration file which corresponded with processed data.

In the next part the user chooses the type of information to display. He can choose from the following types (*info_type*):

RESPONSE Response of server (in miliseconds).

FLOW Data flow (in Bytes per second).

FLOW FROM SERVER Data flow from server (in Bytes per second).

FLOW FROM CLIENT Data flow from client (in Bytes per second).

CONNECTIONS PER SECOND Number of connections created during one second.

REQUESTS PER SECOND Number of requests processed during one second.

CONNECT TIME Time of creating connection to given server.

User can choose if he wants to display maximal, average or minimal values (*value_type*) for every type. It is possible to choose two another types **CONNECTIONS** (number of created connections) and **REQUESTS** (number of processed requests) where neither maximal nor average nor minimal values don't make any sense. There has to be chosen some of types above to create query.

In the last part user chooses the time interval for which he wants to display given information. He can choose from the list of standard values or he can directly fill in limit values of requested interval. The data aggregation level (it means how long time interval is aggregated into one value of result) is set automatically according to the length of given interval. The aggregated intervals use to be one level lower then the level of given interval (eg. if user makes query on values from the last hour, the resultant values are aggregated from minute intervals).

The following example shows the whole query:

service :

Name	IP address	Port	data_type	
serv0	212.20.96.20	80	def_tcp	<input type="checkbox"/>
			tcp	<input checked="" type="checkbox"/>
			www	<input type="checkbox"/>
serv1	212.20.96.22	80	www	<input type="checkbox"/>
			tcp	<input checked="" type="checkbox"/>
serv2	212.20.96.23	80	www	<input type="checkbox"/>
			tcp	<input checked="" type="checkbox"/>
serv3	212.20.96.24	80	www	<input type="checkbox"/>
			tcp	<input checked="" type="checkbox"/>

info type : or info type :

value type :

time range :

or

time from:
hour: minute: second:
day: month: year:

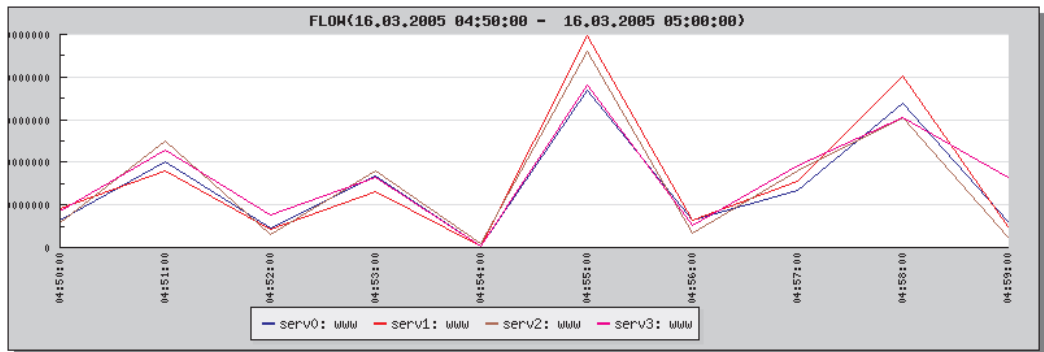
time to:
hour: minute: second:
day: month: year:

Figure 7.3: The complete query

7.2.2 Displaying results of query

The query is sent to statistic module after pushing the Show button and the incoming result is displayed. There is individual line in the graph and individual column in the table for every data type of every server.

The following example shows the result of query on average data flow during 10 minute interval for chosen servers:



FLOW

16.03.2005 04:50:00 - 16.03.2005 05:00:00

	serv0: www	serv1: www	serv2: www	serv3: www
16.03.2005 04:50:00	12911073	18444902	11499486	16829491
16.03.2005 04:51:00	39962627	35780886	50233930	45734776
16.03.2005 04:52:00	9107820	8336780	6304561	15347077
16.03.2005 04:53:00	33411771	26425383	35959374	33017732
16.03.2005 04:54:00	499154	399624	1957623	563161
16.03.2005 04:55:00	73799652	99155361	91903437	76103637
16.03.2005 04:56:00	12691033	13013891	6670939	10574994
16.03.2005 04:57:00	26814288	30933811	35737838	38211602
16.03.2005 04:58:00	67798458	80251549	60826853	61183421
16.03.2005 04:59:00	12361838	9975147	4631462	32650961

Figure 7.4: Result of query

7.3 Configuration

In this part there is possible to change the setting of Load Balancer by doing changes in configuration file and propagate them into the running Load Balancer.

The configuration includes five parts:

- Admin
- View Config
- Main
- Aliases

- Sections

7.3.1 Admin

This section allows work with the individual configuration files, restarting and ending of Load Balancer. Web interface works with three different configuration files:

running the actual Load Balancer's configuration, it is loaded by init module after startup.

default includes the basic Load Balancer's settings.

editing keeps changes done through web interface.

Administration section includes five buttons:

Load Loads the actual running configuration file into the editing one. The default configuration file is loaded instead of the running one if Load Balancer doesn't run.

Load default Loads the default configuration file into the editing one.

Save Saves the editing configuration file into the actual running configuration file. The changes are processed by Load Balancer after its new startup or restart.

Restart Restarts the whole Load Balancer. Init module reads the configuration file and sends restart signal to other modules. They read the new configuration file after their restart.

Shutdown Ends up the whole Load Balancer.

7.3.2 The view of configuration file



In this part there is displayed the editing configuration file in the text form. The changes are displayed in text form after doing them through the web interface. This file is eventually saved as the running configuration file and other parts of Load Balancer use it in this form.





7.3.3 Main

In this part there is possible to change the MAIN section from configuration file. Every record corresponds to one section. The individual records can be deleted by trash icone next to the name of module. If the module includes more sections they can be deleted using trash icone next to their name. There is the possibility to add existing section by choosing one of the list and pushing the add new section button at the end of sections' list for given module.

There is the row under the list of all modules which allows adding of a new module. It is created by pushing the add new module button after submitting of the name and one included section.

MODULES

- **init** 
 - [MODULE_INTT](#)
 - 

- **core** 
 - [MODULE_CORE](#) 
 - [MODULE_PING](#) 
 - 


- Modul: Section: 

Figure 7.5: Editing of MAIN section

There is the table in the bottom which informes about ALIASES section. It is divided into two columns. There is the list of active aliases (which are used in the MAIN section) in the left part. All chosen aliases are moved to the group of inactive after pushing the Disable button.

There is the list of inactive aliases in the right part of the table. All chosen aliases are moved to the group of active after pushing the Enable button.

7.3.4 Aliases

In this part there is possible to change the ALIASES section from the configuration file.

ALIASES

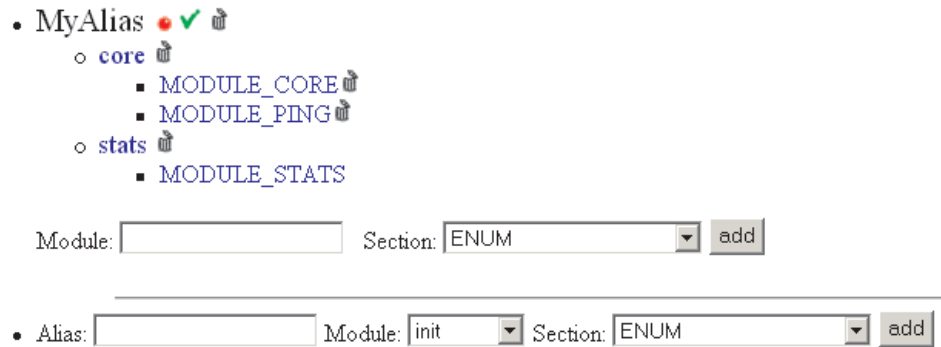


Figure 7.6: Editing of ALIASES section

There are three icons next to the name of each alias. The first informs about the state of alias (the green button means active alias, the red button means inactive alias). The second icon allows activation or deactivation of alias and the third trash icon deletes the alias.

There is displayed the list of modules under the name which are being substituted in the MAIN section. There is the trash icon next to the name of module for deleting it. There is the local list of sections for every module (the same as for modules in the MAIN section). If the module includes more sections they can be deleted using trash icon next to their name.

There is the part with adding of a new section into the alias in the end of module's list for each alias. The section is added into the module if it already exists. Otherwise the new module is created with one new section. The section is added after pushing the add button.

There is the part allowing the adding of new alias under the list of all aliases. It is necessary to fill in the name, module and section and push the add button.

7.3.5 Sections

The list of all sections from the configuration file is displayed in the main page. The sections which are not included in any module or section has the trash icon next to their name for their deleting. There is a part which allows adding of a new section under the list of sections. The new section is added after submitting the name and pushing the add new section button.

There are displayed individual variables, enumeration types and tables from the selected section in the lower part of page after pushing the selected

section. Every item is displayed in the individual frame. There is the data type in brackets next to the name of item. The trash icon next to the data type allows to delete given item from the section.

The enumeration types should be defined in the ENUM section which is reserved for these types, but they can be defined even in user sections.

log_level (defenum)

(if you can delete leave field empty)

Value: Value: Value:

Value: Value: Value:

Value: Value:

Add new:

(use comma for add more new values - value1,value2,value3)

Figure 7.7: Editing of enumeration type

The list of values of enumeration type is displayed in the top. The value is changed by its editation and pushing the change button. If the new value is empty string it is removed from the enumeration type. The new value is added by Add new edit window and pushing the change button. Several values are separated by commas.

INIT_STATS_STARTUP_TIMEOUT (varint)

Value:

(Stats sartup timeout in sec.)


INIT_SYSLOG_LEVEL (varstring)

(Set level of messages written to syslog in all modules.)

Figure 7.8: Editing of variable

If the variable has any range defined its value is displayed as the expanding list of values. Its change is done by choosing another value and pushing the change button.

If the variable has no range of values its value is displayed in editing window. Its change is done by direct editation and pushing the change button.

SERVERS (tabledef) 


	Name	IP protocol	IP address	IP port	Static weight
	my server	TCP	192.168.0.3	80	100
	<input type="text"/>	TCP	<input type="text"/>	<input type="text"/>	<input type="text"/>

Figure 7.9: Editing of table

The first row of the table displays the names of individual columns, then there are the individual rows with data and the last row is empty and serves for submitting of new values. The change of values is done similarly as for variables by direct editation or choosing of new value and pushing the change button. The new value adding is done by filling of the last row and pushing the change button.

There is the Add button on the end of the page which allows adding of new items. After pushing this button displays the new page where are three frames for adding of new variable, table or enumeration type.

New variable

Choose type:

Write name:

Write value:

Write range:

Write short comment:

Write long comment:

In range you can use list of values or emms separated by comma.

Figure 7.10: Adding of the new variable

The data type and name are needed to be put in when adding new variable. The other optional values are the variable's value, value range, short comment and long comment. The new variable is added by pushing the Add button.

The screenshot shows a form titled "New table". It contains the following fields and controls:

- "Write name:" with a text input field containing "MyNewTable".
- "Write number of columns:" with a text input field containing "5".
- "Write comment:" with a large text area containing "This is my brand new table. I will use to store my own values." and a vertical scrollbar on the right.
- A "Next" button located at the bottom center of the form.

Figure 7.11: Adding of the new table - first step

The name and number of columns are needed to be put in for adding the new table. The optional value is the comment for the whole table. The next step for creating table is displayed after pushing the Next button.

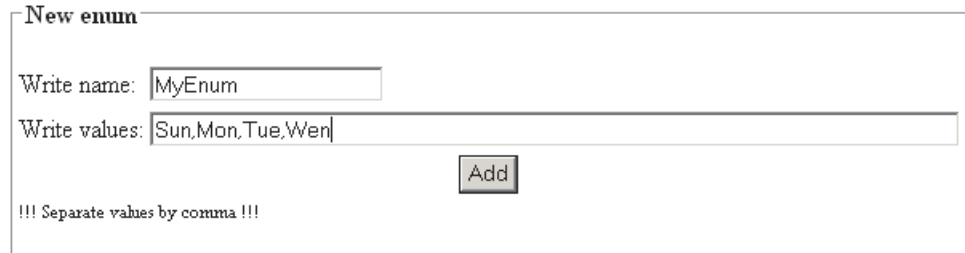
The screenshot shows a form titled "MyNewTable" with the following sections:

- "Write name of columns to next row.": A row of five text input fields containing "First", "Second", "Third", "Fourth", and "Fifth".
- "Write type of columns to next row.": A row of five dropdown menus with the following options: "varint", "vardouble", "varchar", "varbool", and "varservers".
- "Write range of columns to next row.": A row of five text input fields. The fourth field contains "#bool".
- "Write comment of columns to next row.": A row of five empty text input fields.
- An "Add" button.
- A long text area at the bottom containing "This is my brand new table. I will use to store my own values.".

Figure 7.12: Adding of the new table - second step

There is necessary to correctly set data types for individual columns in the second step of creating table. These data types are chosen from the list of values in the second row, the other parameters are optional. There is possible to fill in the names of individual columns in the first row, ranges of

values in the third row and comments in the fourth row. The table is added by pushing the Add button.



The image shows a web form titled "New enum". It contains two input fields: "Write name:" with the value "MyEnum" and "Write values:" with the value "Sun,Mon,Tue,Wen". Below the input fields is a button labeled "Add". At the bottom of the form, there is a note: "!!! Separate values by comma !!!".

Figure 7.13: Adding of the new enumeration type

The name and the list of values are needed to be put in for adding the new enumeration type. The individual values are separated by commas. The new enumeration type is added by pushing the Add button.

Chapter 8

Submodules

There are described individual balancing algorithms in this chapter which can be used by Load Balancer. How to add another algorithm or how to change existing one can be found in the programmers documentation.

8.1 First layer library

The *lib_layer1_main.so* library with *common* function is the part of Load Balancer. This function conforms to the rules from the *L1_RULES* table in *<LAYER1_SUBMODULE>* section in configuration file. The common function goes through the table and compares parameters of actual connection to current rule in the table. The rule is used when it corresponds to all parameters.

The rules is made up of 14 columns:

name The name of rule. Its name is added to data type identifier string which is sent to statistic module if this rule is used by Load Balancer.

stats Information about current connection are sent to statistic module if this parameter is set up to true value.

protocol The list of protocols to which is current rule related. The rule is ignored if no protocol is included.

destination port The port of service where listens the core module (eg. 80 for HTTP). The rule matches any port of given protocols if this value equals 0.

source port The port from which connected the client. The rule matches any client port if this value equals 0.

source IP address

source IP address mask IP address of client is compared to IP address and mask in the rule. The rule matches if at least one of the following conditions is fulfilled:

- The IP address is empty or equals 0 after converting it to number value
- The mask of IP address is empty or equals 0 after converting it to number value and the source IP address equals IP address in rule at the same time.
- The source IP address is included in the range of addresses defined by IP address and its mask in the rule.

second layer library

second layer function The pair of parameters determines the library and function on the second layer. Given function is called after the common function termination if both of parameters are valid and this rule is used.

third layer library

third layer function The pair of parameters determines the library and function on the third layer. Given function is called after the common function termination if both of parameters are valid, this rule is used and no valid library and function was applied on the second layer.

data layer library

data layer function The pair of parameters determines the library and function on the data layer. Given function is called in the core module when the function in data layer submodule is called if both of parameters are valid and this rule is used.

servers The identifier of server or group of servers. The priority of all servers which matches to given value is set to 1 if the rule is matched. It is chosen among these servers on other layers. The rule is ignored if no server matches given value.

8.2 Second layer library

The *http_layer2.so* library with *http_headers* function is the part of Load Balancer. It uses parameters from the section `<HTTP_SUBMODULE>` in configuration file for setting of this function. This function is used for HTTP protocol balancing on the application layer according to RFC2616 standard. The function reads the HTTP header and according to its data it distributes connections. Reading of HTTP header is limited by the following parameters:

MAX_URI The maximal length of URI.

MAX_URI_HEADER_BODY The maximal length of record body in the header.

MAX_HEADERS The maximal number of records in header.

The *http_headers* function receives individual parts of header and it compares them to the values in rules. If there is not set the retrieval in header records there can be used the following values of "where to apply" parameter in the table of rules:

default The rule is used no matter the values of other parameters.

method The method of protocol (eg. GET, POST, ...).

request URI URI received in the header.

HTTP version The version of HTTP protocol.

scheme The scheme of request (eg. http).

host The host of request (IP address of server). The address of "Host" record is used when there is such a record in the header.

port The destination port of the request.

absolute path The absolute path of request on the server.

query The query of request, it includes the beginning question mark.

The "http" value is appended if no schema is included in the request header. The 80 value is appended if no port is included in the request header. The "/" value is appended if no absolute path is included in the request header.

If there is set the retrieval in header records there is used the header record which has the same name as the rule.

The list of rules in *HTTP_RULES* is made up of 14 columns:

name The name of rule. Its name is added to data type identifier string which is sent to statistic module if this rule is used by Load Balancer.

stats Information about current connection are sent to statistic module if this parameter is set up to true value.

compare in headers The string is searched in header record specified by parameter "where to apply" if this parameter is set up. The string is searched in the header if this parameter is not set up.

where to apply It includes the name of header if there is set up searching in the headers records. It can include one of "default", "method", "request URI", "HTTP version", "scheme", "host", "port", "absolute path", "query" values if there is not set up searching in the headers records.

regular expression The searched string is considered to be regular expression if this parameter is set up to true.

case sensitive The searched string is case-sensitive if this parameter is set up to true.

string The string value which is compared to the request according to rules mentioned above.

second layer library

second layer function The pair of parameters determines the library and function on the second layer. Given function is called after the common function termination if both of parameters are valid and this rule is used.

third layer library

third layer function The pair of parameters determines the library and function on the third layer. Given function is called after the common function termination if both of parameters are valid, this rule is used and no valid library and function was applied on the second layer.

data layer library

data layer function The pair of parameters determines the library and function on the data layer. Given function is called in the core module when the function in data layer submodule is called if both of parameters are valid and this rule is used.

servers The identifier of server or group of servers. The priority of all servers which matches to given value is set to 1 if the rule is matched. It is chosen among these servers on other layers. The rule is ignored if no server matches given value.

The priority value is kept the same for the servers which corresponds to the rule if the rule is used. The priority value is set up to 0 for other servers.

Other function of this submodule is the usage of cache which is used when the variable *USE_CACHE* is set up to be true.

Cache includes the list of rows where everyone correspondes to one hash number. Every row includes entries with the same hash number (conflict domain). There is saved hash string, server and timestamp in every entry. There is saved number of server in the cache where was the last connection redirected. The conflict domain given by hash number is checked through after submodule query and the function returns number of server and timestamp if the requested record is found.

The *CACHE_SIZE* parameter determines the number of rows in the cache, the *CACHE_DOMAIN_SIZE* parameter determines the number of entries in conflict domain and the *CACHE_STRING_SIZE* parameter determines the maximal length of hash string.

The string in parameter *CACHE_WHERE_APPLY* is used for caching. The values of string can include "method", "request URI", "HTTP version", "scheme", "host", "port", "absolute path", "query" which means the hash string will be taken from chosen part of HTTP header. Hash string will gain value which corresponds to chosen rule if the value of parameter string equals "rule match".

There is added the *CACHE_PRIORITY* value to the server's priority if the cache returns one of chosen servers and cache record is not older than *CACHE_AGE* seconds.

8.3 Third layer library

The *static_layer3.so* library with static algorithms and *dynamic_layer3.so* library with dynamic algorithms are the parts of Load Balancer. The static algorithms use static values for choosing appropriate server, the dynamic algorithms use queries to statistic modul.

8.3.1 Static algorithms

Basic round robin

The basic algorithm which remembers the last chosen server and sets the servers' priority to use the last chosen server as the last from chosen group. This algorithm is not suitable for the farm configuration where any group of servers overlap themselves. It chooses servers in the same way as DNS round-robin algorithm if all servers are in the same group. This algorithm is one of the fastest and it is suitable as DNS round-robin replacement.

Random select

This algorithm randomly chooses the destination server from all servers having the positive priority.

Hash IP select

This algorithm chooses the destination server according to source IP address. The resultant number of address is divided by number of server having positive priority. The division remainder determines the index among this servers.

Self weighted round robin

This algorithm extends the basic round robin algorithm to more groups of servers. It saves number of connections for every server and it chooses the server with the lowest number of connections among servers having positive priority. If there are more servers having the lowest number of connections it chooses the server with the lowest index. This algorithm is suitable to use for more groups of servers which don't overlap and the round robin algorithm would be used for each of the groups.

Static select first

This algorithm uses static weights of individual servers. It should be used for farm of servers which have the different performance. Their performance is set by static weight in configuration file and this algorithm balances equally according to given static weights. It can be used even for groups of servers which overlap themselves.

It saves number of connections for every server. It counts actual weight as $actual = \frac{weight}{static_weight * priority}$ for every server having positive priority where weight is the number of connections redirected to given server, the static_weight is static weight of server from configuration file and the priority

is the priority of server from preceding modules. The algorithm chooses the server having the lowest actual weight value.

8.3.2 Dynamic algorithms

The dynamic algorithms use for decisions data saved in statistic module. User can set if queries for this data should be done during processing of every request or this data should be actualized only once in given time. This setting can be done in *L3_QUERY_RATES* table which is included in *<LAYER3_SUBMODULE>* section of configuration file:

rule Name of the rule from *LAYER1_SUBMODULE* which will use given values. Given values will be used by all rules which are not included in this table if name is empty. For all such rules are queries done during processing of every request otherwise.

number of requests The new query for values actualization is done after processing of given count of requests.

time between queries The new query for values actualization is done after given time in milliseconds.

Minimal server response

This algorithm balances chosen group of servers according to their maximal response. The highest priority is set up to the server which has the lowest maximal response during the last second.

Minimal average server response

This algorithm balances chosen group of servers according to their average response. The highest priority is set up to the server which has the lowest average response during the last second. It improves the previous algorithm in the case that response of some server has higher variance than other ones.

Least open connections

This algorithm balances chosen group of servers according to their actually open connections count. The highest priority is set up to the server which has the lowest number of actually open connections.

Weighted least open connections

It improves the previous algorithm by the possibility of different server performance settings. This algorithm balances chosen group of servers according to their actually open connections count and static weight. The highest priority is set up to the server which has the lowest ratio of actually open connections count to static weight.

Weighted minimal sessions count

This algorithm balances chosen group of servers according to their open connections count during last second and static weight. The highest priority is set up to the server which has the lowest ratio of open connections count during last second to static weight.

Weighted minimal flow from server

This algorithm balances chosen group of servers according to their data flow from server during last second and static weight. The highest priority is set up to the server which has the lowest ratio of data flow from server during last second to static weight.

Predictive minimal server response

This algorithm balances chosen group of servers according to the last progress of their maximal response. The more is decreasing the server response the more connections can be redirected there and so the higher priority the server gets. The servers are compared according to the number $c = \frac{r_{10}}{r_5} * \frac{r_5}{r_1}$, where r_{10} is the maximal server response from the last ten responses, r_5 is the maximal server response from the last five responses and r_1 is the last server response. The highest priority is set up to the server which has the highest resultant number c .

Predictive minimal flow from server

This algorithm balances chosen group of servers according to the last progress of their average data flow from server. The more is decreasing the data flow from server the more connections can be redirected there and so the higher priority the server gets. The servers are compared according to the number $f = \frac{f_{10}}{f_5} * \frac{f_5}{f_1}$, where f_{10} is the average data flow from server during last ten seconds, f_5 is the average data flow from server during last five seconds

and f_1 is the average data flow from server during last second. The highest priority is set up to the server which has the highest resultant number f .

Predictive minimal sessions count per second

This algorithm balances chosen group of servers according to the last progress of their average connections count per second. The more is decreasing the average connections count per second the more connections can be redirected there and so the higher priority the server gets. The servers are compared according to the number $c = \frac{c_{10}}{c_5} * \frac{c_5}{c_1}$, where c_{10} is the average connections count per second during last ten seconds, c_5 is the average connections count per second during last five seconds and c_1 is the connections count per the last second. The highest priority is set up to the server which has the highest resultant number c .

8.4 Data layer library

The *data_layer.so* library with *ftp* function is the part of Load Balancer. This function is set up as submodule on data layer during the calling of submodules on the first to third layer. The core module then calls the function in specific periods during connection processing. It changes incoming and outgoing data and opens data connections to make ftp protocol functional in both active and passive modes.