# Frequently Asked Questions

## Table of contents

## Questions

## 1. Basic Terminology

### 1.1. What is the batch?

Batch is a group of clients with the same characteristics and loading behavior. Configuration file has one (normally) or more batches.

### 1.2. What means UAS?

UAS - user activity simulation - fetching url, sleeping, another url -sleeping cycles, simulating actual user activity.

## 2. Hardware Requirements

### 2.1. What are the minimal HW requirements?

A PC with Pentium-III and 200 MB may be used for loads with hundreds simultaneous loading clients.

### 2.2. What is the recommended HW configuration?

Note, that each virtual client takes about 30-35 K memory in user-space plus some none-pageable memory in kernel mainly for send and recv buffers.
Our PC with Pentium-4 2.4 GHz and 480 MB memory is capable to support 3000-4000 simultaneously loading clients. To reach 10K simultaneously loading clients a PC with dual 3.0 GHz Intel or 2.2 GHz AMD and 1-2 GB memory is recommended, but for some configurations a single high-end CPU like Xeon or Opteron could be enough.
When loading from a multi-CPU (or multi-core) systems, a loading model with the number of curl-loader processes equal to the number of CPUs may be most effective and deliver even higher numbers (than 10K) of simultaneously loading clients from a single PC. If you are thinking, tuning, optimizing and getting our advise - the sky is the limit.

## 3. Building curl-loader

### 3.1. Which operating systems are supported by curl-loader?

You can use any operating system of your choice, providing this it is linux with kernel 2.4 or

2.6.

### 3.2. What are the building pre-requirements?

General C development environment with bash, gcc, make, etc on a linux machine is required.
Building pre-requirements are:
1. openssl binaries and openssl development package with include files (on debian libssl-dev);
2. ncurses binaries and development package (on debian e.g. libncurses5-dev).
Adjust Makefile variables to point to the openssl headers and libraries. To specify an openssl development directory with include files (e.g. crypto.h), export environment variable OPENSSLDIR with the value of that directory.
For example:

```
$export OPENSSLDIR=the-full-path-to-the-directory
```

Another known issue is libidn.so, which means, that some linux distributions do have some libidn.so.11, but not libidn.so. Resolve it by creating a softlink.
Tarball of curl is included to the current release. When libcurl or curl-loader have building issues, correct them in the Makefile.

### 3.3. How to make (build) curl-loader?

Run the following commands from your bash linux shell:

```
$tar zxfv curl-loader-<version>.tar.gz
$cd curl-loader-<version>
$make
```

By default, we are building both libcurl and curl-loader without optimization and with debugging -g option. To build with optimization and without debugging, please, run:

```
$make cleanall
$make optimize=1 debug=0
```

If still any building issues, please, fill you free to contact us for assistance using placed in download tarball PROBLEM-REPORTING form and its instructions.

## 4. Creating Loading Configuration

### 4.1. How can I create loading configuration file?

To run a load create your configuration file to be passed to curl-loader using -f commmand line option, e.g.

```
#curl-loader -f ./conf-user/user_batch.conf
```

For more examples, please, look at the files in "conf-examples" directory. You may copy an example file and edit it by using the next FAQ guidelines.

Yet another option to start is by running "$make menuconfig" configuration GUI. The dialog window will guide you and create your configuration file in conf-user directory. A limitation of the menu-guided configuration, is that it enables to create only a single UAS URL.
If you need more that a single UAS url, copy your batch file from the conf-user directory to some other place and edit it by adding as much url tag series (below) as you need, but don't forget to update UAS_URLS_NUM accordingly.
The UAS url tag series:
UAS_URL=
UAS_URL_USERNAME=
UAS_URL_PASSWORD=
UAS_URL_MAX_TIME =
UAS_URL_INTERLEAVE_TIME =
For more details you may look at the next FAQ guidelines.

## 4.2. What are the loading configuration file tags and semantics?

Configuration file or "batch configuration file" consists from tag=value strings, groupped into 4 sections:
- General;
- Login;
- UAS (user activity simulation by fetching urls and waiting for timeouts);
- Logoff;

Configuration file should possess at least one client batch defined with the following params in each batch:

```
---------------------------------------------------------------------------
########### GENERAL SECTION ##################################
BATCH_NAME= bulk_batch # The name of the batch. Logfile - bulk_batch.log
CLIENTS_NUM=300 # Number of clients in the batch
CLIENTS_INITIAL_INC=30 # Clients to be added each second till CLIENTS_NUM
INTERFACE = eth0 # Name of the network interface from which to load
NETMASK=255.255.240.0 # Netmask either as an IPv4 dotted string or as a CIDR number
IP_ADDR_MIN= 192.168.1.1 # Client addresses range starting address
IP_ADDR_MAX= 192.168.5.255 # Client addresses range last address
CYCLES_NUM= 100 # Number of loading cycles to run, -1 means forever
USER_AGENT="" # User-Agent HTTP header quoted string.
#When empty string or missed - MSIE-6 default
CUSTOM_HEADER="My-header1: header1" # Up to 16 custom headers.
#
```

```
########### LOGIN SECTION ####################################
LOGIN=n # If 'y' or 'Y', login enabled, all other tags of the
# section to be filled. If 'n' or 'N' - comment out others
#LOGIN_USERNAME= # Redundant tag, use LOGIN_URL_USERNAME instead
#LOGIN_PASSWORD= # Redundant tag, use LOGIN_URL_PASSWORD instead
#LOGIN_REQ_TYPE= # To be either GET+POST, POST or GET. Depricated.
# Use instead LOGIN_REQ_TYPE_GET_POST, LOGIN_REQ_TYPE_POST
# or LOGIN_REQ_TYPE_GET tags by using tag=y to enable
#
#LOGIN_CREDENTIALS_FILE=""# A file with strings <user><separator><password>
#LOGIN_POST_STR= # POST string matrix. See below.
#LOGIN_URL= # A valid http or https url to be used for login
#LOGIN_URL_USERNAME="" # Username for login url
#LOGIN_URL_PASSWORD="" # Password for login url
#LOGIN_URL_MAX_TIME= # Maximum batch time in seconds to login
#LOGIN_URL_INTERLEAVE_TIME= # Time in msec to sleep after login
#LOGIN_CYCLING= # If 'y' login should be run in cycles, and not
# just done only once
########### UAS SECTION ######################################
UAS=y # If 'y' or 'Y', login enabled, and other lines of the section to be filled
UAS_URLS_NUM = 2 # Number of urls
UAS_URL=ftp://anonymous:stam@localhost/curl-7.16.0.tar.gz # Rather large file
#UAS_URL_USERNAME="" # Username for this particular UAS url
#UAS_URL_PASSWORD="" # Password for this particular UAS url
UAS_URL_MAX_TIME = 20 # Maximum batch time in seconds to fetch the url
UAS_URL_INTERLEAVE_TIME = 0 # Time in msec to sleep after fetching the url
UAS_URL= http://localhost/apache2-default/index.html
#UAS_URL_USERNAME="" # Username for this particular UAS url
#UAS_URL_PASSWORD="" # Password for this particular UAS url
UAS_URL_MAX_TIME = 4 # Maximum batch time in seconds to fetch the url
UAS_URL_INTERLEAVE_TIME = 0 # Time in msec to sleep after fetching the url
# You may add any number of urls providing 5-tags for each url as above,
# but do not forget to update the UAS_URLS_NUM.
########### LOGOFF SECTION ###################################
LOGOFF=n # If 'y' or 'Y', login enabled, and other tags of the
# section to be filled. If 'n' or 'N' - comment out others
#LOGOFF_REQ_TYPE= # To be GET, GET+POST, or POST. Depricated.
# Use instead LOGOFF_REQ_TYPE_GET_POST, LOGOFF_REQ_TYPE_POST
# or LOGOFF_REQ_TYPE_GET tags by using tag=y to enable
#LOGOFF_POST_STR= # String to be used for logoff, like "op=logoff"
```

```
#LOGOFF_URL= # A valid http or https url to be used for logoff
#LOGOFF_URL_USERNAME="" # Username for logoff url
#LOGOFF_URL_PASSWORD="" # Password for logoff url
#LOGOFF_URL_MAX_TIME= # Maximum batch time in seconds to logoff
#LOGOFF_URL_INTERLEAVE_TIME= # Time in msec to sleep after logoff
#LOGOFF_CYCLING= # If 'y' login should be run in cycles, and not just done only once
```

Worth to mention, that each batch configuration should contain all tags from the section GENERAL as well as LOGIN, UAS and LOGOFF section tags.
When LOGIN, UAS or LOGOFF is set as 'y' (yes), all tags for that section should appear (uncommented) and to be set to some valid values (empty string "" to be used not to define LOGIN_POST_STR and LOGOFF_POST_STR tags, when POST is not applicable).
In the case, that LOGIN, UAS or LOGOFF section tag is disabled by setting 'n' (no) value, thus, all the tags of the disabled section may appear with with empty strings "", without values or may be commented out by '#'.

Tag LOGIN_POST_STR:
To generate multiple unique users with unique passwords, use the string like "user=%s%d&password=%s%d". First '%s' will be substituted by the value of LOGIN_USERNAME tag and '%d' by the client number. Second '%s' will be substituted by LOGIN_PASSWORD tag value and second '%d' by the same client number. For example, if LOGIN_USERNAME=robert, LOGIN_PASSWORD=stam and LOGIN_POST_STR "user=%s%d&password=%s%d", the final POST string, used for the client number 1, will be "user=robert1 password;=stam1".
In this case LOGIN_USERNAME and LOGIN_PASSWORD strings are used just as base-words for generating unique user credentials by appending an number.
To use the username and password 'as as', provide LOGIN_POST_STR without %d symbols, e.g. "user=%s&secret=%s". Thus, all clients will have the same POST credentials with the string looking like "user=robert& secret=stam".
The option without %d symbols to be used also, when LOGIN_CREDENTIALS_FILE is defined,
because the credentials uniqueness is ensured by the file content.
Another allowed syntax serves to generate unique user, all using with the same password: "user=%s%d&secret=%s".
Note, that the words like 'username', 'user', 'password', 'secret', etc are those fields, that login users are required to fill in their POST page.

Tag CLIENTS_INITIAL_INC:
serves to increase clients number gradually at the initial phase of loading. Use the tag in GENERAL section to specify number of loading clients to be added each second till the total number reaches the number specified by CLIENTS_NUM tag.

Tag CUSTOM_HEADER:
is assisting to customize/add/over-write HTTP/FTP headers. Up to 16 custom HTTP/FTP headers are allowed. If a header already exits by default, the custom header over-writes it. USER_AGENT tag is for User-Agent header only, whereas by CUSTOM_HEADER may be added or over-written any header (including User-Agent). An example of batch configuration is ./conf-examples/custom_hdrs.conf.

Tag LOGIN_CREDENTIALS_FILE:
specifies the path to the file with credentials (full-path or relative to curl-loader). A text file with usernames and passwords with the structure of each string like: <user><separator><password>can be used as an input. According to RFC1738, only reserved ':', '@', '/' and probably ' ' (space) are safe to use as separators between username and password. An example of batch configuration is ./conf-examples/credentials_fr_file.conf and an example of credentials is in ./conf-examples/credentials.cred.

Note, that both quotted and non-quotted string are supported as the tags values.
For more examples, please, look at the files in "conf-examples" directory.

## 4.3. How does the configuration support login, logoff and authentication flavors?

curl-loader performs login and logoff operations using the following HTTP methods:
- GET+POST (server response to GET provides a post-form to be filled and posted by POST);
- POST only;
- GET only.

Both UAS and Login URLs are coming with an option to configure username and password. The difference is that for UAS only GET method is currently allowed in and username&password are intended to support Web or Proxy Authentication (see below). Login and Logoff urls allow POSTing user credentials via configurable POST forms as well as configuring a unique username and password for each virtual client by appending a sequence number to the username and password basewords. When Login url fetching is challenged by Web or Proxy Authentication it will use user credentials specified/generated as described.
Does somebody really needs PUT or POST methods as an option for UAS urls?
If yes, please, signal it to us.
The loader supports HTTP Web Authentication and Proxy Authentication. The supported authentication methods are Basic, Digest (RFC2617) and NTLM. When responded 401 or 407, libcurl will choose the most safe method from those, supported by the server.
To support GSS Web-Authentication, add in Makefile building of libcurl against appropriate GSS library, see libcurl pages for detailed instructions.

### 4.4. When is better to use Login/Logoff section and when UAS?

Login and Logoff section enable usage of POST-ed credentials, whereas UAS urls are using only GET method.
Another business case for Login url is that it may be either used either in cycling or in non-cycling mode, whereas UAS urls are always cycled.
When e.g. proxy authentication is the first operation to be performed and only once, non-cycling Login url may be useful.

## 5. What about FTP load?

To generate FTP/FTPS load, please, use UAS section and pass user credentials via ftp-url according to the RFC 1738 like:
ftp://username:password@hostname:port/etc-str
Please, look at conf-examples/ftp.conf

## 6. Running Load

### 6.1. What are the running environment requirements?

Running hundred and thousand of clients, please, do not forget:
- to increase limit of descriptors (sockets) by running e.g.
#ulimit -n 19999;
- optionally, to set reuse of sockets in time-wait state: by setting
#echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle and/or
#echo 1 > /proc/sys/net/ipv4/tcp_tw_reuse;
In some cases you may need to increase the system limits for open descriptors (sockets):
echo 65535 > /proc/sys/fs/file-max
Note, that your linux distribution may require editing of some other files like /etc/security/limits.conf, etc

### 6.2. How I can run the load?

Usage: run as a root user:
#./curl-loader -f <configuration filename> [other options]
Other possible options are:
-c[onnection establishment timeout, seconds]
-e[rror drop client (smooth mode). Client on error doesn't attempt loading any more]
-h[elp]
-i[ntermediate (snapshot) statistics time interval (default 3 sec)]

-f[ilename of configuration to run (batches of clients)]
-l[ogfile max size in MB (default 1024). On the size reached, file pointer rewinded]
-m[ode of loading, 0 - hyper, 1 - storming, 2 - smooth (the default)]
-o[utput to stdout bodies of downloaded files - attn!- bulky]
-r[euse connections disabled. Closes TCP-connections and re-open them. Try with and without]
-s[tderr printout of client messages instead of to logfile - attn!- bulky]
-t[hreads enable - enables threads, each runs a batch of clients]
-v[erbose output to the logfiles; includes info about headers sent/received]
-u[rl logging - logs url names to logfile, when -v verbose option is used]
For the rare cases, when several batches are specified in the same config file, note, please, that curl-loader without -t runs only the first batch. Thus, running several client batches without threads requires some script starting several curl-loader processes.
Connection Reuse Disable Option (-r):
The default behavior of curl-loader after HTTP response is to re-use the tcp-connection for the next request. If you are specifying -r command-line option, the TCP connection will be closed and re-opened for the next request. Whether it is appropriate for you to work with -r option or without, it depends on your server support of Keep-Alive and the purpose of your testing. Try with and without -r and see, what you get.
Connection reuse (which is the default, without -r option) has advantages due to the decreased consumption of opened descriptors (sockets) and ports.

## 6.3. Which loading modes are supported?

The default loading mode is a so-called "smooth" (by default or -m2 command line), where each client spends as much time as wishes and starts another url or another cycle only after completing the previous url. When a client loading is terminated due to some reason, e.g. connection timeout, the default behavior is to schedule loading for this client at the next cycle. If -e option is passed to command line, the client will not be re-scheduled any more, which is usefull to get indication of errors by monitoring drop in number of active clients. Look in the logfile for errors, and when connection timeout error appears, adjust the connection timeout using -c command line option. Note, that the smooth mode is suitable for login-logoff cycles of load.
Hyper-mode (command line -m0) is currently consider as an experimental, but steadily finds more and more usage, particularly for heavy loads. The mode is basically the same as the smooth mode, but is using for event demultiplexing epoll() or /dev/epoll instead of poll(). The mode is promising to deliver more loading clients from a single thread.
Another loading mode is called "storming" (-m1 command line), where all clients of a batch are starting together. They are expected to finish their job within a certain timeout. After the timeout the clients either accomplished fetching url, or are cutted and disconnected; thus, a new loading cycle begins. Storming mode is better to use, when login operation is done once

and not in cycles, or when the tool is used just as a traffic generator or to generate "storming" bursts of requests.

## 6.4. How I can monitor loading progress status?

curl-loader outputs to the console loading status and statistics as the "standing" ouput, where the upper part is for the latest interval and below is the average numbers since load start. The example is here:

```
=============================================================
Last interval stats (interval:3 sec, clients:10, CAPS:3):
Operations: Success Failed
LOGIN: 4 1
UAS-0: 3 0
LOGOFF: 2 0
HTTP/FTP-Req:10,2xx:5,3xx:4,4xx:0,5xx:1,Err:0,D:20(ms),D-2xx:12(ms),T-in:12015(B/s),T-out:35511(B/
HTTPS/FTPS-Req:0,2xx:0,3xx:0,4xx:0,5xx:0,Err:0,D:0(ms),D-2xx:0(ms),T-in:0(B/s),T-out:0(B/s)
------------------------------------------------------
Summary stats since load start (load runs:24 secs, CAPS-average:4):
Operations: Success Failed
LOGIN: 0 32
UAS-0: 0 0
LOGOFF: 0 0
HTTP/FTP-Req:95,2xx:47,3xx:48,4xx:0,5xx:3,Err:1,D:31(ms),D-2xx:15(ms),T-in:113921(B/s),T-out:33021
HTTPS/FTPS-Req:0,2xx:0,3xx:0,4xx:0,5xx:0,Err:0,D:0(ms),D-2xx:0(ms),T-in:0(B/s),T-out:0(B/s)
=============================================================
```

A copy of the output is also saved in the file <batch-name>.txt

## 6.5. Where is the detaled log of all virtual clients activities and how to read it?

DETAILED LOGFILE is written to the file named:
<batch-name>.log:
The semantics of logfile output, using command line options -v (verbous) and -u (url print):
"Cycle number", "Client number (ip-address)" - some information string, e.g.:

```
4 39 (192.168.0.39) :== Info: Trying 10.30.6.42... : eff-url:
http://10.30.6.42:8888/server/Admin/ServiceList.do, url:
```

Which meas: cycle: 4, client number 39 with ipv4 address (192.168.0.39), status of the message is Info, eff-url - is the url, used right now, "url:" is empty, which means, that it is the same as effective.
Effective url may be a result of redirection and, thus, "url:"
(target url, specified in batch configuration file) will be printed as well.
Please, note, that when the logfile reaches 1024 MB size, curl-loader rewinds it and starts to

overwrite it from the beginning. Y may tune the rewinding file size by using command line option:
-l <log-filesize-in-MB>

## 6.6. Which statistics is collected and how to get to it?

Currently HTTP/HTTPS statistics includes the following counters:
- requests num;
- 2xx success num;
- 3xx redirects num;
- client 4xx errors num;
- server 5xx errors num;
- other errors num, like resolving, tcp-connect, server closing or empty responses number;
- average application server Delay (msec), estimated as the time between HTTP request and HTTP response without taking into the account network latency (RTT);
- average application server Delay for 2xx (success) HTTP-responses, as above, but only for 2xx responses. The motivation for that is that 3xx redirections and 5xx server errors/rejects may not necessarily provide a true indication of a testing server working functionality.
- throughput out (batch average);
- throughput in (batch average);

The statistics goes to the screen (both the interval and the current summary statistics for the load) as well as to the file with name <batch_name>.txt When the load completes or when the user presses CTRL-C (sometimes some clients may stall), the final load report is printed at the console as well as to the statistics file.
Some strings from the file:

```
--------------------------------------------------------------------------------------------------------------------------------------
Run-Time,Appl,Clients,Req,2xx,3xx,4xx,5xx,Err,Delay,Delay-2xx,Thr-In,Thr-Out
2, HTTP/FTP, 100, 155, 0, 96, 0, 0, 1154, 1154, 2108414, 15538
2, HTTPS/FTPS, 100, 0, 0, 0, 0, 0, 0, 0, 0, 0
4, HTTP/FTP, 100, 75, 32, 69, 0, 0, 1267, 1559, 1634656, 8181
4, HTTPS/FTPS, 100, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cutted here
36, HTTP/FTP, 39, 98, 35, 58, 0, 0, 869, 851, 1339168, 11392
36, HTTPS/FTPS, 39, 0, 0, 0, 0, 0, 0, 0, 0, 0
38, HTTP/FTP, 3, 91, 44, 62, 0, 0, 530, 587, 1353899, 10136
38, HTTPS/FTPS, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0
*, *, *, *, *, *, *, *, *, *, *, *
Run-Time,Appl,Clients,Req,2xx,3xx,4xx,5xx,Err,Delay,Delay-2xx,Thr-In,Thr-Out
38, HTTP/FTP, 0, 2050, 643, 1407, 0, 213, 725, 812, 1610688, 11706
38, HTTPS/FTPS, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
--------------------------------------------------------------------------------------------------------------------------------------
```

The bottom strings after asterisks are for final averages.
At the same time a clients dump file with name <batch_name>.ctx is generated to provide detailed statistics about each client state and statistics counters.
One string from the file:

```
1 (192.168.0.1)
,cycles:124,cstate:1,b-in:22722029,b-out:174605,req:745,2xx:497,3xx:248,4xx:0,5xx:0,err:0
```

where
1 (192.168.0.1)- is the index of the client and its ip-address;
cycles- number of loading cycles done;
cstate - is the number of the client state (-1 - error, 0 - init, 1- login, 2- uas, 3-logoff, 4-final-ok);
b-in - bytes passed in;
b-out - bytes passed out;
req- number of requests done;
2xx, 3xx, 4xx, 5xx - number of responses Nxx received;
err - number of libcurl errors at the resolving, TCP/IP and TLS/SSL levels;
The following conditions are considered as errors:
- error at the level of libcurl, which includes resolving, TCP/IP and, when applicable, TLS/SSL errors;
- all HTTP 5xx server errors;
- most of HTTP 4xx client errors, excluding 401 and 407 authentication responses not considered real errors;

When the above error conditions occur, a virtual client is marked as being in the error state. By default we "recover" such client by scheduling it to the next loading cycle, starting from the first operation of the cycle. You may use command line option -e to change the default behavior to another, so that clients once arriving at error state will not be scheduled for load any more.

## 7. Advanced Issues

### 7.1. What about performance?

HYPER mode (-m 0 command line option) was already used for loads with up to 10000 clients, whereas HW issues, like CPU and memory resource, start to be counting. The mode uses epoll() syscall (via libevent library) for demultiplexing.

### 7.2. How to run a really big load?

0. Try the default smooth and hyper (experimental) modes and see, what is more appropriate for you. Smooth mode is stable and tested well. Hyper is less restrictive about the allowed

number of clients, but still requires more development and optimization.
Every big load starts with a small load. First, see, that you have a working configuration file. Run it with a 1-2-3 clients and commandline option -v, and look into the l<batch-name>.log logfile.
Look into the HW issues, duscussed in the FAQs and mind, that each client requires 30-35 K of memory.
1. Compile with optimization;
Since you need performance compile with optimization and without debugging.

```
$make cleanall
$make optimize=1 debug=0
```

Y may add to Makefile optimization for your particular processor by -match /-mcpu gcc option directives to OPT_FLAGS.
2. Login as a su;
3. Increase the default number of allowed open descriptors (sockets);
Run e.g. #ulimit -n 19900
When running several instances of curl-loader, consider increase of system limits for open descriptors, if necessary. Take your own account of the socket usage in the system, considering sockets faster recycling (less time in the time-wait state), by setting, optionally, something like this:

```
#echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle and/or
#echo 1 > /proc/sys/net/ipv4/tcp_tw_reuse;
```

Correct, if required, the value of CURL_LOADER_FD_SETSIZE (set to 20000 in Makefile) to control the maximum fd, that may be used for select. This is not required to be cared about for hyper mode.
Increase the maximum number of open descriptors in your linux system, if required, using linux HOWTOS.

```
echo 65535 > /proc/sys/fs/file-max
```

Note, that your linux distribution may require editing of some other files like /etc/security/limits.conf, etc
4. Relax routing checks;
Relax routing checks for your loading network interface. When "eth0" used for loading run:

```
echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter
echo 0 > /proc/sys/net/ipv4/conf/eth0/rp_filter
```

5. Increase memory available for kernel tcp;
Read the maximum available TCP memory and sysctl or echo as a root the number to the kernel tcp, e.g.:

```
cat /proc/sys/net/core/wmem_max - the output is 109568.
/sbin/sysctl net.ipv4.tcp_mem="109568 109568 109568" or
echo "109568 109568 109568" > /proc/sys/net/ipv4/tcp_mem
```

6. Create configuration files for each instance of curl-loader to run.

What is important is to give a unique value for tag BATCH_NAME, which is in use by a separate instance of curl-loader. Logfile, report file, etc have name, which are the derivatives of the BATCH_NAME value. Therefore, when several instances of curl-loader are writing to the same file, this is not helpful and may be even "crashful". Please, use in your configuration batch files non-overlapping ranges of IP-addresses, else libcurl virtual clients will compete for the IP-addresses to bind to.

Use CLIENTS_INITIAL_INC tag in smooth or hyper mode to increase number of your clients gradually at start-up in order not boom the server and not to burn out the CPU at your loading machine. Addition of new clients is a CPU-expensive operation, therefore keep CLIENTS_INITIAL_INC below 100-200 clients per second.

7. Connections re-use.

The default behavior of curl-loader now is to re-use the tcp-connection for the next request. This default decreases consumption of CPU and open sockets. If you are specifying -r command-line option, the connection will be closed and re-opened for the next request.

8. Troubleshooting.

Run the first loading attempt with a small number of clients using command-line options -v (verbose) and -u (url in logs). Grep to look for the errors and their reasons. If an error is "Connection timeout", you may try to increase the connection establishment timeout (the default is 5 seconds), using -c command-line option.

If any assistance required, please, don't hesitate to contact us using PROBLEM-REPORTING form in the download tarball.

9. Logs and statistics.

After end of a run, or after SIGINT (Cntl-C), the final results are calculated and printed to the console as well as to the file <batch-name>.txt. Current results are presented in each row, and average summary as the last raws, separated from the rest by asterisks.

Pay attention, that <batch-name>.log log file may become huge, particularly, when using verbose output (-v -u). Command-line option -l <maxsize in MB> may be useful, whereas the default policy is to rewind the logfile (writing from the file start), when it reaches 1 GB. Do not use -v and -u options, when you have performance issues.

10. Monitoring of the loading PC.

Please, use top or vmstat commands to monitor the memory usage, swapping and CPU. Intensive swapping is a good indication, that your PC is short in user-space memory. If you see "memory pressure" counters in netstat -s output, this is a good indication, that the PC is short in kernel memory for TCP.

Zero idle CPU is not the show stopper, but when you see, that Load Status GUI on your console prints its output with delays higher than it should be (the default is 3 seconds and may be adjusted by -i command-line option), you need stronger CPU or run several curl-loader processes on a multi-CPU machine. Note, that for a load with several curl-loader precesses you need to arrange different configuration files with different batch-names and not overlapping ranges of IP-addresses for each curl-loader process.

## 7.3. How to calculate CAPS numbers for a load?

When number of clients is defined by CLIENTS_NUM tag, number of CAPS (call attempts per seconds) is resulting from the clients number and load duration for each cycle, comprising from:
- login time with possible redirections and sleeping after login interval;
- uas time for each url with possible redirections, intervals betwee urls and after uas interval;
- logoff time with possible redirections and sleeping after logoff interval;

The actions and time intervals are configurable in batch file, whereas url retrival time is server and network dependent and not always easy to predict. The result is that number of clients/requests is a known parameter, and number of CAPS is something to be estimated from the time of test and number of requests.
Smooth and hyper modes are presenting at the LOAD STATUS GUI the output of calculated current and average CAPS.