Frequently Asked Questions

Table of contents

Questions

1. Basic Terminology

1.1. What is the batch?

Batch is a group of clients (virtual clients emulating real clients) with the same characteristics and loading behavior. Meaning of the word "batch" is also overloaded and is used for a batch configuration file, representing a test plan.

1.2. What is the batch configuration file?

Batch configuration file is a test plan for a batch of clients.

1.3. For which version are these FAQs applicable?

We are keeping these FAQs updated and applicable for the latest version of curl-loader.

1.4. What is the maximum possible number of virtual clients (VCs) with a dedicated IP-address?

You can use the same IP-address for all VCs or a dedicated address for each client. A clear limitation of "the same IP" is that you can reach only 64K of clients (64K is the number of ports to bind).
When using a dedicated IP address per client, the address is added by curl-loader to the loading network interface as a secondary IP. Talking 32-bits, linux kernel allocates for IPv4 structure in_ifaddr, which is is ~55-60 bytes (depending on config options, arch, etc) from slab-64, which means, that something above 64 bytes are used. IPv6 address takes a chunk from slab-128 and uses above 128 bytes of kernel memory.
We got at least 100K IPv4 and 8K IPv6 addresses using a machine with 1G memory. The more memory you have, the more you can get, whereas it depends on the number of chunks in slab-64 and slab-128 allocators.
The situation can be worse for 64-bit kernels, whereas such machines are normally enjoying more memory.

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 we would recommend a PC with a single 3.2 GHz Intel or 2.2 GHz AMD and 1.5-2 GB of memory.
When loading from a multi-CPU (or multi-core) systems, a loading model with the number of curl-loader threads equal to the number of logical CPUs seen in /proc/cpuinfo (command-line option -t <thread-num>) may be the most effective and deliver very high numbers (20K-50K-100K) of simultaneously loading clients from a single PC. To run a batch in several threads use the command line option -t <number-of-threads>.
If you are thinking, tuning, optimizing and getting our advise, thus, 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 that this is linux with kernel 2.4 or 2.6. We are building, developing and testing on Debian and kernel 2.6 is recommended. If you have any compilation issues on any other linux distributions, please, report them to the mailing list.

3.2. What are the building pre-requirements?

General C development environment with bash, gcc, make, etc on a linux machine is required. We are supporting gcc of series 3 and 4.
The Building pre-requirements are:
1. glibc include files, which are normally in /usr/include;
2. openssl binaries;
3. openssl development package with include files (on debian libssl-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 or in some cases exclude -lidn

from the linking string, which requires commenting a string in Makefile and uncommenting another.
Tarball of curl is included to the current release. When libcurl or curl-loader has 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
Y can optionally install the loader by running as a root:
#make install
and enjoy our man pages by $man curl-loader and $man curl-loader-config.
A known building issue is libidn.so, which means, that some linux distributions do have some libidn.so.11 or libidn.so.10, but not libidn.so, and it breaks linking. Resolve the issue by creating a softlink:
#cd to the directory, where e.g. libidn.so.11 is installed and run
#ln -s libidn.so.11 libidn.so
If the above was not helpful, try editing the Makefile and removing -lidn from the linking string.
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 command line option, e.g.
#curl-loader -f ./conf-examples/bulk.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.

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

Configuration file or "batch configuration file" consists from tag=value strings, grouped into

the 2 sections:
- General;
- URLs;

Section General contains common for the batch parameters, whereas section URLs has one or more URL subsections each starting with a certain URL and containing more tags.
An example of a simple configuration file to be used for traffic generation client side against e.g. lighttpd web server: --

```
########### GENERAL SECTION ##################
BATCH_NAME= 10K-clients
CLIENTS_NUM_MAX=10000
CLIENTS_NUM_START=100
CLIENTS_RAMPUP_INC=50
INTERFACE =eth0
NETMASK=255.255.0.0
IP_ADDR_MIN= 192.168.1.1
IP_ADDR_MAX= 192.168.53.255
CYCLES_NUM= -1
URLS_NUM= 1
########### URLs SECTION #######################
URL=http://localhost/index.html
URL_SHORT_NAME="local-index"
REQUEST_TYPE=GET
TIMER_URL_COMPLETION = 0
TIMER_AFTER_URL_SLEEP = 0
```

The name of the batch is "10K-clients" with the maximum number of clients to be 10000; load will be started with initial 100 clients and each seconds will be added 50 more clients. Network interface eth0 will be used for the load with unique IP-address for each client taken from 192.168.1.1 up to 192.168.53.255 with netmask 255.255.0.0. Number of cycles is to be performed by each client in not limited (CYCLES_NUM= -1) and load will proceed till user presses Cntl-C to stop it. Each client fetches a single url (URLS_NUM= 1) http://localhost/index.html.
The url was given a short name "local-index", and this name will appear at the Load Status GUI. The url will be fetched using HTTP GET request method and the download will not be limited by time (TIMER_URL_COMPLETION = 0). If we would like to limit this operation by time, we could place here some number in milli-seconds, e.g. TIMER_URL_COMPLETION=3000; the timer will be monitored, enforced and reported, if missed, at Load Status GUI and other statistics. By default all missed url-completion timers are considered as errors and added to statistics as T-err. Each client after accomplishing a url, including possible protocol redirections, waits/sleeps for 0 ms (TIMER_AFTER_URL_SLEEP = 0, which means, that actually client does not wait/sleep)

and goes to the next URL, if any, to fetch.
Yet a one more example of a configuration file used for login to a web-service, perform some activity and logoff, all in cycles, is presented below:

```
########### GENERAL SECTION ###################
#
BATCH_NAME=web-service
CLIENTS_NUM_MAX=100
CLIENTS_RAMPUP_INC=2
INTERFACE=eth0
NETMASK=24
IP_ADDR_MIN=192.168.0.2
IP_ADDR_MAX=192.168.0.2
URLS_NUM=4
CYCLES_NUM= 200
########### URLs SECTION ######################
### Login URL - cycling
# GET-part
#
URL=http://192.168.0.1:8888/vax/root/Admin
URL_SHORT_NAME="Login-GET"
REQUEST_TYPE=GET
TIMER_URL_COMPLETION = 10000
TIMER_AFTER_URL_SLEEP =5000
# POST-part
#
URL=""
URL_SHORT_NAME="Login-POST"
URL_USE_CURRENT=1
USERNAME= admin
PASSWORD= admin-passphrase
REQUEST_TYPE=POST
FORM_USAGE_TYPE= SINGLE_USER
FORM_STRING= username=%s&password=%s
TIMER_URL_COMPLETION = 4000
TIMER_AFTER_URL_SLEEP =1000
### Cycling URL
#
URL=http://192.168.0.1:8888/vax/Admin/ServiceList.do
URL_SHORT_NAME="Service List"
REQUEST_TYPE=GET
```

```
TIMER_URL_COMPLETION = 3000
TIMER_AFTER_URL_SLEEP =1000
### Logoff URL - cycling, uses GET and cookies to logoff.
#
URL=http://192.168.0.1:8888/vax/Admin/Logout.do
URL_SHORT_NAME="Logoff"
REQUEST_TYPE=GET
TIMER_URL_COMPLETION = 0
TIMER_AFTER_URL_SLEEP =1000
```

The name of the batch is "web-service", maximum number of clients 100, load will started with initial 2 clients and each seconds will be added 2 more clients.

Eth0 will be used with the same IP-address for each client 192.168.0.2 with netmask 255.255.255.0 (24 in CIDR notation). If we would like to assign to each client a unique IP, it could be done by providing a range of at least 100 IP-address, e.g. IP_ADDR_MIN=192.168.0.10 and IP_ADDR_MAX=192.168.0.110. Number of cycles is to be performed by each client is limited to 200, and the load stops, when each client accomplishes 200 cycles. A client fetches up to 4 urls, specified in section URLs

The first URL operation, which is doing a simulated user/client, is login. Login is performed by GET-ing the page (with 3xx HTTP redirections handled inside) with the first url http://192.168.0.1:8888/vax/root/Admin, called shortly "Login-GET". Maximum time considered appropriate for this operation is 10000 msecs. The 2000 msec "sleep" timer simulates here the time taken by a user to fill username and password to the POST-form received by the GET.

The second url is empty and is used for POST-ing to the "current" url, using the form. Such empty string urls are allowed only, when URL_USE_CURRENT=1 is specified. The url is named as "Login-POST". Username "admin" and password "admin-passphrase" are used as the single user type and "filled" to the FORM_STRING= username=%s&password=%s. The resulting form, which is sent to the web-service, is "username=admin&password=admin-passphrase". There are other options available for the tag FORM_STRING, such as unique generated users, unique generated passwords or credentials loaded from a file; these options will be described later. The maximum time allowed for this POST operation is 4000 msecs, and the time of waiting/sleeping till the next url is 1000 msecs. Worth to mention, that when the authentication is accomplished successfully, server sets to client cookies and redirects the client to the service page, which is the next url (note, that all redirections are handled internally).

The third url is http://192.168.0.1:8888/vax/Admin/ServiceList.do with a short name "Service List". After the url is fetched a client comes to the forth url http://192.168.0.1:8888/vax/Admin/Logout.do for logoff. This batch (testing plan) does not control and limit time for logoff as indicated by TIMER_URL_COMPLETION = 0. After 1000 msecs sleeping a client comes back to the first url to repeat the sequence till the number

of cycles is expired.
If we would like, that login and logoff to be done for each client only once, we would add to the first, second and forth url the tag URL_DONT_CYCLE=1 to mark the urls as not-cycled to be fetched only once.
Note, that both quoted and non-quoted strings are supported as the tags values.
For more examples, please, look at the files in "conf-examples" directory.

4.3. What are the exact meanings of all these tags?

Tags of the sections GENERAL:
BATCH_NAME requires a string value; it is used to name the batch. The name is displayed, while the program is running, and as the base word for the three generated log and statistics filenames.
CLIENTS_NUM_MAX - is the maximum number of clients to be used for this load. Any positive number is valid here. Note, that each loading client requires about 30 K of RAM plus some non-pageable memory in kernel. We have already tried loads with up to 100 K clients. Loads above 1000 client is recommended to start with a some lower starting number, managed by the next tag CLIENTS_NUM_START and ramped-up each second by adding CLIENTS_RAMPUP_INC clients to the loading set.
CLIENTS_NUM_START - is the number of clients to start the load with. If the tag is absent or zero, the value of CLIENTS_RAMPUP_INC will be taken instead. If neither CLIENTS_NUM_START nor CLIENTS_RAMPUP_INC are possessing a valid positive value, the loader takes CLIENTS_NUM_MAX as the number of clients to start with.
CLIENTS_RAMPUP_INC - number of clients to be added to the load in the auto mode every second till CLIENTS_NUM_MAX number will be reached. For machines with a single CPU we would recommend not to keep the number above 50-100, whereas for dual-CPU machines you may try as large as 200-300.
INTERFACE - name of the loading network interface, like eth0. Find the network interface names by running /sbin/ifconfig.
NETMASK - netmask for the loading IP-addresses of virtual clients (from IP_ADDR_MIN to IP_ADDR_MAX). For IPv4 you can use either quad-dotted netmask string or CIDR number from 0-32. For IPv6 only CIDR values from 0 to 128 are supported.
IP-addresses from IP_ADDR_MIN up to IP_ADDR_MAX are the interval of addresses to be used for loading clients. The interval of addresses should be large enough to supply a dedicated IP for each client. Another mode that can be used is "single IP address" for all clients. When IP_ADDR_MIN and IP_ADDR_MAX have the same valid IP-address, it will be used for all clients. If the IP-addresses specified by the tags do not exist, curl-loader adds them as the secondary IPs to the loading network interface.
Remember, that you should ensure smooth routing from the addresses used to server and from server back to the client addresses. Another route to ensure, when is appropriate, is to/from DNS server for resolving. Firewalling (netfilter/iptables) settings are also among

those to check.
IP_SHARED_NUM is the positive number of IP-addresses, that will be assigned by the program to its virtual clients evenly. When the number is one, it is equivalent to the single common IP-address, described in the IP_ADDR_MIN tag section. If, for example, IP_SHARED_NUM is 3, the first client gets the first address, second the second, third gets the third IP-address, whereas the forth client gets back the first address and so on. This option enables to overcome the limit of 64K clients per IP without using a dedicated IP-address for each client. Without the tag the program follows the behavior described for tag IP_ADDR_MIN.
CYCLES_NUM is the number of cycles to be performed. Any positive value is valid here as well as (-1), which means cycle indefinitely. Loading can be normally stopped at any moment by pressing Cntl-C; all statistics will be collected properly and files will be closed. Note, that cycling relates only to the URLs without tag URL_DONT_CYCLE = 1.
Curl-loader enables two optional non-cycling areas: the first area before the cycling url/s area and the second area after the cycling url/s area.
USER_AGENT provides an option to over-write the default MSIE-6-like HTTP header User-Agent. Place here a quoted string to emulate the browser that you need. The header is entered globally. If you need an option to customize it on a per-URL bases, please, use the tag HEADER="User-Agent: the string of my favorite browser".
URLS_NUM is the number of urls to be used in URLs Section.
Tags of the section URLs:
URL - is the first tag of each url-subsection; it marks the beginning of a new url. The tag should be either a valid url or an empty string, which is allowed only when URL_USE_CURRENT=1. "Current url" is useful, for example, when POST-ing page with a url, resulting from the previous GET. HTTP redirections are handled internally, and there is no need to use "current url" approach for that purpose. Note, that URL used for file uploading should include a filename and not only a directory.
URL_SHORT_NAME - is an optional up to 12 ASCII string long name, which will appear at a Load Status GUI. It improves appearance of the current operational status at the console. We are recommending to configure it.
URL_USE_CURRENT - is used in combination with an empty string URL="" to indicate, that the current url page, fetched by the previous operation, should be used. "Current url" is useful e.g., when POST-ing page of a url, resulting from the previous GET. Note, that HTTP redirections are handled internally, and there is no need to use "current url" approach for that purpose.
URL_DONT_CYCLE - marks a url as non-cycling. Curl-loader enables two optional non-cycling areas: the first area before the cycling url/s area and the second after it. The first non-cycling area may be useful for login, authentication, etc. single time operations, whereas the second non-cycling area may be good for a single time logoff, for example.
REQUEST_TYPE - HTTP request method to be chosen from GET, POST or PUT.

UPLOAD_FILE - a filename, including path, of a file to be used for uploading. The path should be taken from the curl-loader place, e.g. ./conf-examples/bax.conf
HEADER - is assisting to customize/add/over-write HTTP/FTP headers. If a header already exits by default, the custom header over-writes it. USER_AGENT tag is for User-Agent header only, whereas by HEADER may be added or over-written any header, including User-Agent. To prevent from sending "Expect: 100-continue", please, pass HEADER="Expect: "
Look at the example ./conf-examples/custom_hdrs.conf.
USERNAME and PASSWORD are the tags to provide credentials for login operations. The strings may be used either as is or as the base-words to append numbers and to generate some unique credentials. The behavior is governed by the tag FORM_USAGE_TYPE. Note, that PASSWORD may be an empty string.
FORM_USAGE_TYPE governs user login process.
"UNIQUE_USERS_AND_PASSWORDS" is used to generate unique usernames and passwords by appending client sequence numbers (starting from 1) to USERNAME and PASSWORD tags within FORM_STRING template.
"UNIQUE_USERS_SAME_PASSWORD" to be used when generating unique users with the same password. "SINGLE_USER" is useful, when all clients are making login using the same USERNAME and PASSWORD. "RECORDS_FROM_FILE" means, that user credentials to be loaded from the file specified by FORM_RECORDS_FILE value. "AS_IS" means, that the FORM_STRING template to be used just AS IS and without any usernames or passwords either from the relevant tags or from file.
FORM_STRING is the configurable template form to be used for POST-ing credentials or any other records.
To generate multiple unique users with unique passwords FORM_USAGE_TYPE= "UNIQUE_USERS_AND_PASSWORDS", and the FORM_STRING to be a template like "username=%s%d&password=%s%d". First '%s' will be substituted by the value of USERNAME tag and '%d' by the client number. Second '%s' will be substituted by PASSWORD tag value and second '%d' by the same client number. For example, if USERNAME=robert, PASSWORD=stam and FORM_STRING "username=%s%d&password=%s%d", the final POST string, used for the client number 1, will be "username=robert1&password=stam1".
In this case USERNAME and PASSWORD strings are used just as base-words for generating unique user credentials by appending a client sequence number.
When FORM_USAGE_TYPE="UNIQUE_USERS_SAME_PASSWORD" template of FORM string to be something like "username=%s%d&password=%s", where only username will be unique using the same approach as described above.
When FORM_USAGE_TYPE="SINGLE_USER", provide FORM_STRING 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".

When FORM_USAGE_TYPE= "RECORDS_FROM_FILE" FORM_STRING to be also without %d symbols, because the credentials uniqueness is supposed to be ensured by the file content.
FORM_USAGE_TYPE="AS_IS" means, that FORM_STRING will not be validated and POST-ed indeed AS IS.
FORM_RECORDS_FILE specifies path to the file with credentials or any other tokens. (full-path or relative to curl-loader location). A text file with usernames and passwords with the structure of each string like: <teken1><separator><token2> 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/post-form-token-fr-file.conf and an example of credentials is in ./conf-examples/credentials.cred. Current version supports up to 16 tokens in the file.
FORM_RECORDS_FILE_MAX_NUM allows to load from a file with records, specified by tag FORM_RECORDS_FILE, any number of records, and not only the default number of records, which is equal to the maximum number of virtual clients CLIENTS_NUM_MAX. The tag is required by FORM_RECORDS_RANDOM.
FORM_RECORDS_RANDOM allows to use for each virtual client a randomly chosen record, whereas without the tag the program uses for a client with index I always record number I. One can load e.g. 10000 records from a file, specified by FORM_RECORDS_FILE tag, and use the records in a random fashion for e.g. 100 clients. The tag does not ensure uniquickness of the records used for each virtual client. To use the tag properly, please, specify number of the records to be loaded, using tag FORM_RECORDS_FILE_MAX_NUM.
WEB_AUTH_METHOD to be configured from the "BASIC", "DIGEST", "GSS_NEGOTIATE", "NTLM" or "ANY". The configured method will be used for authentication on HTTP 401 response. When "ANY" is configured, libcurl will choose a method. To use "GSS_NEGOTIATE" the libcurl should be re-compiled with support for GSS.
WEB_AUTH_CREDENTIALS to be provided in the form "user:password". If not configured, the loader uses USERNAME and PASSWORD tags value to synthesize the credentials.
PROXY_AUTH_METHOD to be configured from the "BASIC", "DIGEST", "GSS_NEGOTIATE", "NTLM" or "ANY". The configured method will be used for authentication on HTTP 407 response. When "ANY" is configured, libcurl will choose a method. To use "GSS_NEGOTIATE" the libcurl should be re-compiled with support for GSS.
PROXY_AUTH_CREDENTIALS to be provided in the form "user:password". If not configured, the loader uses USERNAME and PASSWORD tags value to synthesize the credentials.
FRESH_CONNECT is used to define on a per url bases, whether the connection should be

re-used or closed and re-connected after request-response cycle. When 1, the TCP connection will be re-established. The system default is to keep the connection and re-use it as much as server and protocol allow it. Still the system default could be changed by the command-line option -r.
TIMER_TCP_CONN_SETUP is the time in seconds for DNS resolving and TCP connection setup on a per url bases. The global default is 5 seconds, which can be changed using -c command-line option.
TIMER_URL_COMPLETION is the time in milli-seconds to allow a url fetching operation, including all internal re-directions. The legal values are 0, which means no limit on url-completion time, or above 20 (milli-seconds). When a value of the tag is above 0, we are monitoring the progress of url-fetching and canceling it, if not accompliched within the the specified value of milliseconds. The results are presented at the Load Status GUI as the operation "Timed Out" statistics and logged to all statistics files as T-Err number.
TIMER_AFTER_URL_SLEEP is the time in milli-seconds for client to sleep after fetching a URL prior to dealing with the next URL. Zero (0) means don't wait and schedule client immediately. Random timer values are a legal option to be specified as e.g. 0-2000, which means, that a client will sleep for some random time from 0 to 2000 milliseconds.
FTP_ACTIVE, when defined as 1, is forcing FTP protocol to use an active mode (the default is passive).
LOG_RESP_HEADERS, when defined as 1, the program logs response headers to a file. Directory <batch-name> is created with subdirs url0, url1, url<n> Headers of responses are logged to the files named: cl-<client-num>-cycle-<cycle-num>.hdr
LOG_RESP_BODIES, when defined as 1, the program logs response headers to a file. Directory <batch-name> is created with subdirs url0, url1... url<n> Headers of responses are logged to the files named: cl-<client-num>-cycle-<cycle-num>.body
RESPONSE_STATUS_ERRORS supports changes to the default set of per-url responses considered as errors. By default most 4xx (without 401 and 407) and all 5xx response codes are treated as errors. Now you can either add a status to the errors set or remove it. Sign + (plus) adds, - (minus) removes. The syntax is a line of tokens separated by ',', where each tokens begins with either + or - and is followed by the response status number from 0 up to 600.
For example, the effect of RESPONSE_STATUS_ERRORS="+200,-404" is that 200 responses will be considered for that url as errors, whereas 404 will be considered as success.
MULTIPART_FORM_DATA tag adds initial support for multipart form data POST-ing as in RFC1867.
Several tags MULTIPART_FORM_DATA can be used for a url to post, e.g. as in ./conf-examples/multipart-formdata-post.conf:
MULTIPART_FORM_DATA="yourname=Michael"
MULTIPART_FORM_DATA="filedescription=Cool text file with cool text inside"
MULTIPART_FORM_DATA="htmlcode=<HTML></HTML>;type=text/html"

MULTIPART_FORM_DATA="file=@cooltext.txt"
MULTIPART_FORM_DATA="coolfiles=@fil1.gif,fil2.txt,fil3.html"
The files to be uploaded are indicated by @ and to be located in the same directory as curl-loader. Content type could be added as e.g. ;type=text/html. Currently, the feature uses the strings provided AS_IS and does not generate any unique users, unique passwords or loads tokens from file.
To prevent from sending "Expect: 100-continue", pass as a custom HTTP header HEADER="Expect: "
TRANSFER_LIMIT_RATE - limits client maximum throughput for a url. The value of the tag to be provided as bytes (! not bits) per second.
FETCH_PROBABILITY - allows to fetch a url not as a must, but with a certain run-time probability. The allowed values are in the range from 1 to 100 percents.
FETCH_PROBABILITY_ONCE when set to 1 configures each client to make the decision regarding whether to fetch a URL marked by a FETCH_PROBABILITY or not, to be done only once, namely, at the first cycle.

4.4. How does the loader 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.

The loader also supports multipart form data POST-ing as in RFC1867.
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, unless the batch configuration (test plan) is explicitly indicating the method to be used by the tags WEB_AUTH_METHOD and/or PROXY_AUTH_METHOD.
To support GSS Web-Authentication, add in Makefile building of libcurl against appropriate GSS library, see libcurl pages for detailed instructions.

5. What about FTP load and mixed FTP/HTTP load?

To generate FTP/FTPS load, please, pass user credentials via ftp-url according to the RFC 1738 like: ftp://username:password@hostname:port/etc-str
Tag FTP_ACTIVE serves to switch from the default passive FTP to active. Please, look at the examples in conf-examples directory, e.g. ftp.conf, ftp_http.conf, ftp_upload.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 100 000;
- 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 150 000 > /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]
-d[etailed logging; outputs to logfile headers and bodies of requests/responses. Good for text pages/files]
-e[rror drop client. Client on error doesn't attempt to process the next cycle]
-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 is rewinded]
-m[ode of loading, 0 - hyper (the default, epoll () based), 1 - smooth (select () based)]
-r[euse connections disabled. Closes TCP-connections and re-open them. Try with and without]
-t[hreads number. Use it for high loads and when running at SMP/multi-core HW]
-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]
-w[arnings skip]
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.
Threads number -t option. This option is helpful, when running at a multiple CPUs or multiple core CPU HW. We are recommending to use the option only for loads with 1000 clients and more, where the number of threads is kept about the same as the number of the linux logical CPUs seen by cat /proc/cpuinfo.

6.3. Which loading modes are supported?

Hyper-mode (command line -m0) is the default mode . The mode is using for event demultiplexing epoll() or /dev/epoll.
Another loading mode is called "smooth" (-m1 command line), and it is basically the same as hyper, but uses poll() system call for demultiplexing.

6.4. How I can monitor loading progress status?

curl-loader outputs to the console loading status and statistics as the Load Status GUI output, where the left column is for the latest interval and the right is for the summary numbers since load start. A copy of the output is also saved in the file <batch-name>.txt

6.5. Why I am getting "Connection time-out after 5108 ms" like errors in the log file?

This means, that the connections cannot be established within 5 seconds (which is the default). If some of clients are getting such errors, this means, that the server is overloaded and cannot establish TCP connection within the time. Y may wish to increase the connection timeout globally by e.g. -c 10 or to specify an increased timeout using tag TIMER_TCP_CONN_SETUP=10.
When all your clients have such errors, in most cases this is due either to resolving or routing problems or other network problems. Y can set any addresses to your clients in IP_ADDR_MIN and IP_ADDR_MAX tags, providing, that you ensure a smooth route from client and from server for the addresses you use.
Please, start with a single client and see that it works, than progress further. When you are specifying any addresses here, they are added by curl-loader initialization to the network interface of your choice and may be seen by the command:
#/sbin/ip addr
Command ping (man ping) has an option -I to force the ping to be issued from a certain ip-address. To test, that your routing is OK, you may use, e.g.:
#ping -I <the client address you wish to use> url-target
When it works - it will be OK.
Please, also pay attention to the firewalling/NAT settings at your linux loading machine (netfilter/iptables), at the tested servers and through the whole packets route.
Y may also wish to read a bit about linux routing/networking/iptables and DNS resolving

HOWTOS.

6.6. Where is the detailed log of all virtual clients activities and how to read it?

Detailed log is written to the file named:
<batch-name>.log:
The semantics of logfile output, using command line options -v (verbose) 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 means: 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>

6.7. Which statistics is collected and how to get to it?

Currently HTTP/HTTPS statistics includes the following counters:
- run-time in seconds;
- requests num;
- 1xx success 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 (Err);
- url completion time expiration errors (T-Err);
- average application server Delay (msec), estimated as the time between HTTP request and HTTP response without taking into the account network latency (RTT) (D);
- 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 (D-2xx);
- throughput in, batch average, Bytes/sec (T-In);
- throughput out, batch average, Bytes/sec (T-Out);

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,1xx,2xx,3xx,4xx,5xx,Err,T-Err,D,D-2xx,T-In,T-Out
2, Appl , 100, 155, 0, 0, 96, 0, 0, 0, 0, 1154, 1154, 2108414, 15538
2, Sec-Appl, 100, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
4, Appl, 100, 75, 0, 32, 69, 0, 0, 0, 0, 1267, 1559, 1634656, 8181
4, Sec-Appl, 100, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
Cutted here
36, Appl , 39, 98, 0, 35, 58, 0, 0, 0, 0, 869, 851, 1339168, 11392
36, Sec-Appl, 39, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
38, Appl , 3, 91, 0, 44, 62, 0, 0, 0, 0, 530, 587, 1353899, 10136
38, Sec-Appl, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
*, *, *, *, *, *, *, *, *, *, *, *
Run-Time,Appl,Clients,Req,1xx,2xx,3xx,4xx,5xx,Err,T-Err,D,D-2xx,T-In,T-Out
38, Appl , 0, 2050, 0, 643, 1407, 0, 213, 0, 0, 725, 812, 1610688, 11706
38, Sec-Appl, 0, 0, 0, 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- urls, 2-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?

The loader was already used for loads with up to 60000 HTTP 1.1 clients, whereas HW issues, mainly memory resources, start to be counting.

7.2. How to run a really big load?

0. 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, discussed 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 100 000
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 the default hyper mode.

Increase the maximum number of open descriptors in your linux system, if required, using linux HOWTOS.
echo 100 000 > /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_RAMPUP_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_RAMPUP_INC below 100 clients per second per each CPU/core.
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. Running curl-loader in multiple threads on several CPUs/cores.
SMP-support can be gained by using option -t <threads-num>. We can recommend to use number of threads as the number of logical CPUs seen by cat /proc/cpuinfo. Sometimes curl-loader crashes on start with the option used due to suspected bugs in libevent library. Re-run it optionally changing/decreasing number of threads used. Note, that total statistics is written to the file $batch-name_0.txt, whereas logs are per-thread and are written to the files $batch-name_<thread-num>.log.
9. 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. When all your clients fail to connect with "Connection timeout" error, this may be due to routing or resolving problems.
If any assistance required, please, don't hesitate to contact us using PROBLEM-REPORTING form in the download tarball.
10. 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.
11. 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 processes 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_MAX tag, number of CAPS (call attempts per seconds) is resulting from the clients number and load duration for each cycle, comprising from url time for each url with possible redirections, intervals betwee urls and after url 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.