Modernize and optimize HAProxy configuration

Sébastien Gross / HAProxy Technologies

Agenda

1. Optimization example
2. How to measure optimization
3. Timeouts
4. Practical: ACL lookup
5. Practical: Denying a source IP
6. Practical: Backend selection
7. Stick tables
8. Other optimization

Disclaimer

Do not blindly copy / paste examples.

Always read the RTF(E|W)?M.

Optimization example

Example

How can we optimize this directive?

```
http-request deny unless { req.hdr(host) -m regm -i
'^(?:\d|[01]?\d{2}|2(?:[0-4]\d|5[0-5]))\.(?P<byte>(?<=10\.)(
\d|[01]?\d{2}|2(?:[0-4]\d|5[0-5])))\.\k<byte>.(?:[1-9]|[01]?
\d{2}|2(?:[0-4]\d|5[0-4]))(?::(?:\d{1,4}|[0-5]\d{1,4}|6(?:[0
-4]\d{3}|5(?:[0-4]\d{2}|(?:5(?:[0-2]\d|3[0-5])))))?$' }
```

Document your configuration!

- Use comments to document the configuration logic.
- Explain everything you need to know when the production is down.
- Do not describe what specific directive does unless complex ones.
- Use Peter Norvic's Rule of English Translation (Tutorial on Good Lisp Programming, p. 53-54, https://www.cs.umd.edu/~nau/cmsc421/norvig-lisp-style.pdf).

1. Write your configuration logic in English.
2. Write your configuration.
3. Translate your configuration in English.
4. Compare 1 and 3.

Example

```
# Deny any request wich host header does not match following pattern
# 10.A.A.1-254. An optional port (from 1 to 65535) can be specified
# after a semicolon.

http-request deny unless { req.hdr(host) -m regm -i
'^(?:\d|[01]?\d{2}|2(?:[0-4]\d|5[0-5]))\.(?P<byte>(?<=10\.)(\d|[01]?\d{2}|2(?:[0-4]\
d|5[0-5])))\.\k<byte>.(?:[1-9]|[01]?\d{2}|2(?:[0-4]\d|5[0-4]))(?::(?:\d{1,4}|[0-5]\d
{1,4}|6(?:[0-4]\d{3}|5(?:[0-4]\d{2}|(?:5(?:[0-2]\d|3[0-5])))))?$' }
```

Example (better, even if the goal of this rule isn't)

```
# Deny any request which host header does not match following pattern:
# 10.A.A.1-254. An optional port (from 1 to 65535) can be specified
# after a semicolon.
#
# Using this pattern we ensure that all requests are performed with hostname
# set to our production PODs IP in the 10.A.A.0/24 networks. Check the IPAM
# for further information about IP address assignment.

http-request deny unless { req.hdr(host) -m regm -i
'^(?:\d|[01]?\d{2}|2(?:[0-4]\d|5[0-5]))\.(?P<byte>(?<=10\.)(\d|[01]?\d{2}|2(?:[0-4]\
d|5[0-5])))\.\k<byte>.(?:[1-9]|[01]?\d{2}|2(?:[0-4]\d|5[0-4]))(?::(?:\d{1,4}|[0-5]\d
{1,4}|6(?:[0-4]\d{3}|5(?:[0-4]\d{2}|(?:5(?:[0-2]\d|3[0-5])))))?$' }
```

Who spotted the bug?

Port 0 is allowed!

Spoiler: this one is difficult to (cannot be) optimized!

Optimization is also useful to …

- Help your colleagues or future you to quickly understand the configuration logic.
- Help the support to provide you the best possible answer.
- Increase your service uptime.
- Reduce the CPU time required to process a request (less energy required).
- Reduce the request latency (increase the requests per second ratio).

How to measure a request

Measure a request

- Take a reference to compare your results to.
- Test the worst case scenario.
- Do not test just one request.
- Use tools like h1load (https://github.com/wtarreau/h1load) httpress (https://github.com/yarosla/httpress) or h2load (https://nghttp2.org). Do not mix tools.
- Add **`cpu_ns_avg`** in the logs (value is computed at the very end of the request).
- Disable logs when running heavy loaded benchmark tools.
- Avoid virtual machines.

Timeouts

What are timeouts?

- A timeout is the solution, not the problem.
- Protects the proxy against resources exhaustion.
- Act at several levels (**`client`**, **`connect`**, **`server`**, **`tunnel`**, **`http-request`**, …)
- No one-size-fits-all settings.
- Reported in the standard logs (termination state): **`cD`**, **`cR`**, **`SD`**, **`sH`**, …
- Configured in **`default`**, **`frontend`**, **`backend`** and **`listen`** sections.

How to setup timeouts?

- Best practice: use different values for all timeouts.
- Think of a funnel from the client (highest timeout) to the server (lowest timeout).
- For static content, use low values to connect to local servers (5 to 50ms max), higher for distant ones.
- Try with acceptable values and increase them.
- Check logs for termination states (**`cD`**, **`cR`**, **`SD`**, **`sH`**, …)
- Use **`option redispatch`** if session cookie is used in http.
- Use **`tcp-ut`** in addition to long timeouts.

What is the main problem with those settings?

```
default
    option                  redispatch
    retries                 3
    timeout connect         10s
    timeout client          300s
    timeout server          300s
```

10s with 3 retries = up to 40s to inform the client!

If more than 1 server in the backend, the **`redispatch`** helps to try another one. But still 10s to wait.

Timeouts example

```
default
 option redispatch        # client can’t flush session cookie
 timeout client 51s
 timeout server 49s
 timeout connect 25ms     # Should be enough on a LAN
 timeout http-request 5s # Slowloris protection!

frontend web
 # tcp-ut overrides client timeout if the client disappears.
 bind *:80 tcp-ut 10s
```

Disclaimers

Benchmarks in this workshop

- Results will be different (CPU, distribution, HAProxy version, background tasks, …)
- Benchmarks a bound only on 1 CPU core.
- Absolute value means nothing, only consider variations.
- It’s better to use a real lab with dedicated servers.

Results are given for:

- 1 HAProxy (2.6.6) server and 1 injector. Dedicated bare metal.
- Using 1 core of Intel(R) Xeon(R) CPU E5-1630 v4 @ 3.70GHz.

Example files

- All example files are provided with full comments.
- We try to figure out the solution together.
- We are here to learn not to cheat. Please do not read the solution in advance.

ACL lookup

Block obvious passwords

Context:

- The hosted service is used to reset a user password.
- Service is located under **`/reset-password`**.
- The password is provided as a query parameter (**`password=Secret`**).
- For example sake, we only focus on password, not the user name.

Goal:

- Block the request if password is a common English word.

Example: **`/reset-password?password=Secret`**

Original configuration

```
# this frontend denies all requests having a password parameter
# matching a banned list of passwords from the file
# forbidden-passwords.acl
frontend original
    bind :8001
    http-request deny if { query -m reg -f forbidden-passwords.acl }
```

ACL **`forbidden-passwords.acl`** is a list of regular expressions matching **`password=something`** in the query:

```
(^|&)password=(alabama|admission|articles)(&|$)
[...]
```

Reference measurement

```
# the direct frontend is used to measure the req/s in the most
# favorable case. This would be our reference.
frontend direct
    bind :8000
    # Each http-request has a cost. This simulates a dummy rule which
    # does not filter anything but checks request is a GET.
    http-request deny if !{ method -m str GET }
```

Benchmark

Run HAProxy:

```
haproxy -f haproxy.cfg
```

Run reference benchmark (port **8000**):

```
taskset -c 1 ./h1load -t 1 -c 64 -d 20 -S
'http://198.18.0.102:8000/reset-password?foo=bar&password=NotInList&bar=foo'
```

Run lookup benchmark (port **8001**):

```
taskset -c 1 ./h1load -t 1 -c 64 -d 20 -S
'http://198.18.0.102:8001/reset-password?foo=bar&password=NotInList&bar=foo'
```

Results

Direct: **166k req/s**

Filtered: **162k req/s**

Be careful of `tune.pattern.cache-size`.

```
NO_PATTERN_CACHE=1 haproxy -f haproxy.cfg
```

Direct: **165k req/s**

Filtered: **108k req/s**

Test with different URLs

Now let's try a different request: `/?foo=bar&password=NotInList&bar=foo`.

Results are the same (about 108k req/s). **Why?**

The path should be filtered out: `{ path -m str /reset-password }`

Filter on path (1)

```
frontend optim-01
    bind :8002
    http-request deny if { query -m reg -f
forbidden-passwords.acl } { path -m str /reset-password }
```

Results are not better:

- `/reset-password?foo=bar&password=NotInList&bar=foo`: **108k req/s**
- `/?foo=bar&password=NotInList&bar=foo`: **108k req/s**

WHY?

Order matters!

Filter on path (2)

```
frontend optim-02
    bind :8003
    http-request deny if { path -m str /reset-password } { query -m reg -f forbidden-passwords.acl }
```

Results are better:

- `/reset-password?foo=bar&password=NotInList&bar=foo`: **104k req/s**
- `/?foo=bar&password=NotInList&bar=foo`: **165k req/s**

Can we do better?

Yes we can!

Extract password parameter with `url_param`

```
frontend optim-03
    bind :8004
    http-request deny if { path -m str /reset-password } {
url_param(password) -m reg -f forbidden-passwords-o03.acl }
```

```
(alabama|admission|articles)
[...]
```

Results are better:

- `/reset-password?foo=bar&password=NotInList&bar=foo`: **119k req/s**

Can we do better?

Extract password parameter: words list

```
frontend optim-04
    bind :8005
    http-request deny if { path -m str /reset-password } {
url_param(password) -m reg -f forbidden-passwords-o04.acl }
```

```
alabama
admission
articles
[...]
```

Results are worse:

- `/reset-password?foo=bar&password=NotInList&bar=foo`: **33k req/s**

WHY?

Because the list is not 26 items long but 251.

Use fixed string

```
frontend optim-05
    bind :8006
    http-request deny if { path -m str /reset-password } {
url_param(password) -m str -f forbidden-passwords-o04.acl }
```

Results are better:

- `/reset-password?foo=bar&password=NotInList&bar=foo`: **158k req/s**

Bonus 1: case insensitive

```
frontend bonus-01
    bind :8007
    http-request deny if { path -m str /reset-password } {
url_param(password) -m str -i -f forbidden-passwords-o04.acl
}
```

Results are worse:

- `/reset-password?foo=bar&password=NotInList&bar=foo`: **123k req/s**

Why?

Case insensitive lookup is linear.

Bonus 2: case insensitive (2)

```
frontend bonus-02
    bind :8008
    http-request deny if { path -m str /reset-password } {
url_param(password),lower -m str -f forbidden-passwords-o04.acl }
```

Results are excellent:

- `/reset-password?foo=bar&password=NotInList&bar=foo`: **153k req/s**

There is almost no impact!

- Tree lookup is very efficient
- Only a few comparison (depend on longest pattern)

Bonus 3: 10k-entry list

```
frontend bonus-03
    bind :8009
    http-request deny if { path -m str /reset-password } {
url_param(password),lower -m str -f
google-10000-english-no-swears.txt }
```

Results are excellent:

- `/reset-password?foo=bar&password=NotInList&bar=foo`: **151k req/s**

There is almost no impact! Size does not matter!

As a comparison:

- String insensitive lookup: **13k req/s**
- Regular expression lookup: **1.3k req/s**

Conclusions

- Do not just run 1 request.
- Be careful about the pattern cache.
- Careful if regular expressions worked perfectly on staging (no difference with string lookup if <1.3k req/s).
- Avoid regular expressions if possible.
- Prefer case sensitive (normalized) lookups.

Denying source IP

First approach **`http-request deny`**

Goal: deny access unless client is from **`allowed_network`** (**`10.0.0.0/24`**)

```
frontend http-request
    bind :8000

    # Block all source IP but authorized networks.
    acl allowed_networks src -m ip 10.0.0.0/24
    http-request deny if !allowed_networks

    # Force a connection close
    http-after-response add-header connection close if !allowed_networks
```

How can we optimize this frontend?

Rules processing order

- **`tcp-request connection`**: Immediately after acceptance of a new incoming connection. (Layer 4)
- **`tcp-request session`**: Once a session is validated, after all handshakes have been completed. (Layer 5)
- **`tcp-request content`**: Can access to the request payload. Requires **`tcp-inspect delay`**. (Layer 7)
- **`http-request`**: After parsing the request. (Layer 7)
- **`tcp-response content`**: Can access to the response payload. Requires **`tcp-response-inspect delay`**. (Layer 7)
- **`http-response`**: After parsing the response. (Layer 7)
- **`http-after-response`**: Before sending the response to the client. (Layer 7)

Benchmarking

Difficult to measure since:

- A new TCP connection is made for each request.
- Need to check connections per second instead of requests.
- Inconsistent results when running on one core.
- Need to saturate client.
- Lock congestion happens on the client in SSL context.

Command lines to use (to test http-request):

```
taskset -c 0-3 ./h1load -t 4 -c 128 -d 60 -l -S http://198.18.0.102:8000/
taskset -c 0-3 ./h1load -t 4 -c 128 -d 60 -l -S https://198.18.0.102:8010/
taskset -c 0-3 ./h1load -t 4 -c 128 -d 60 -l -S https://198.18.0.102:8020/
```

Results

	HTTP		HTTPS (RSA key)		HTTPS (ECDSA key)	
Rule	**conn/s**	**CPU**	**conn/s**	**CPU**	**conn/s**	**CPU**
`http-request`	74k	98%	<255	100%	2.4k	100%
`tcp-request content`	123k	99%	<255	100%	2.5k	100%
`tcp-request session`	183k	97%	<255	100%	2.6k	100%
`tcp-request connection`	185k	97%	81k (*)	87%	81k (*)	86%

- RSA: 2048-bit RSA bey.
- EC: ECDSA key.
- SSL comes with overhead on client side a well (context creation, preallocation, etc…).
- Do not compare clear vs SSL connections.

(*) client staturares: values can reach up to 90k reqs on a ll 8 cores:

```
taskset -c 0-7 ./h1load -t 8 -c 128 -d 20 -S 'https://198.18.0.102:8023/'
```

Conclusions

- Each layer has a cost.
- Close connections as soon as possible.
- In SSL context `tcp-request connection deny` saves key computations and CPU time.
- If possible prefer ECDSA key instead of RSA (key computation is on client side).
- Avoid `silent-drop` rule directive if firewalls are used.

Backend selection

Hostname based routing

How can we optimize this proxy?

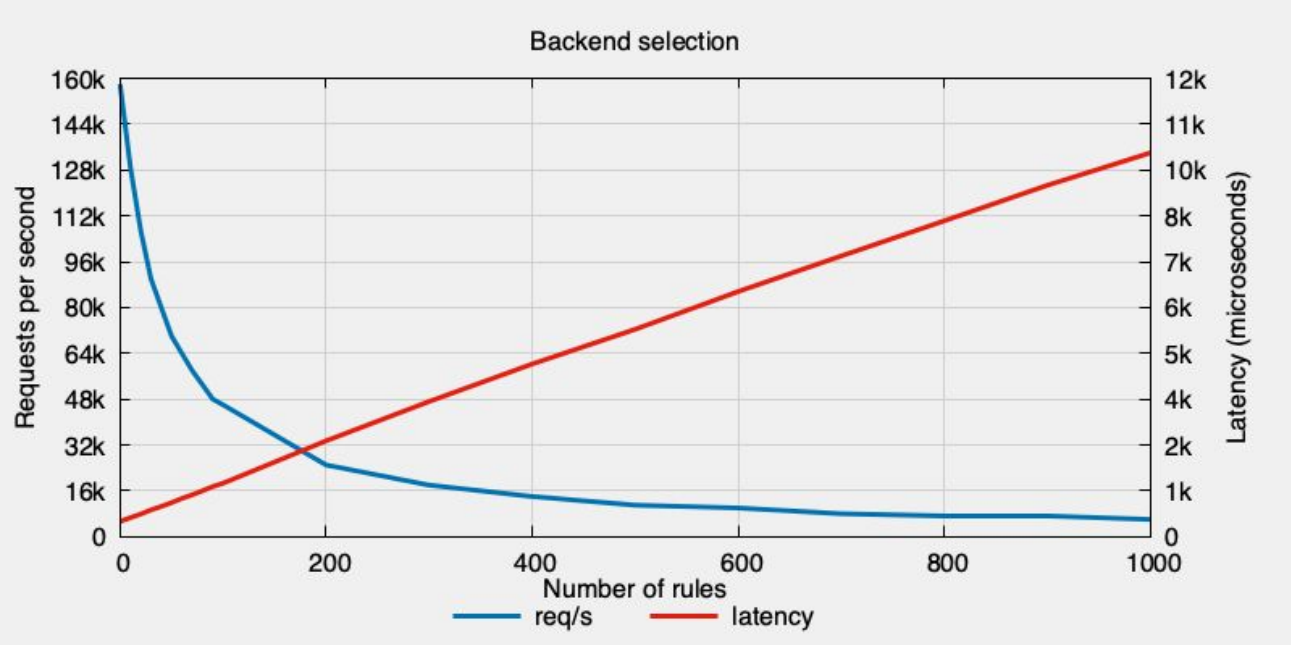
```
# Route traffic to specific backend depending on hostname. "vhost" is
# just a dummy backend given as an example.
frontend original
    bind :8000

    default_backend default

    acl host_vhost00 req.hdr(host) vhost00.example.com
    use_backend vhost if host_vhost00
    [...]
    acl host_vhost99 req.hdr(host) vhost99.example.com
    use_backend vhost if host_vhost99
```

Check how a request is handled

- Backend selection depends on hostname.
- If hostname is unknown, use **`default`** backend.
- Need to run benchmark on several backends:
  - **`vhost00.example.com`** (most favorable case)
  - **`vhost42.example.com`** (random case)
  - **`vhost100.example.com`** (least favorable case)

```
taskset -c 1 ./h1load -t 1 -c 64 -d 20 -l -S -H "host: $vhost" 'http://198.18.0.102:8000/'
```

Results



- Latency increases linary with number of backends.
- Requests per second follow an exponential decay with number of backends.
- Each **`use_backend`** rule's filter has a cost.

Naive approach

Remove named ACL:

```
frontend anonymous-acl
    bind :8001
    use_backend vhost if { req.hdr(host) -m str vhost00.example.com }
    [...]
    use_backend vhost if { req.hdr(host) -m str vhost99.example.com }
```

- Does not change anything (**43k req/s**).
- ACL are evaluated on demand.
- Need a more efficient way to route requests.

Why not a session variable?

Transaction variable

```
frontend anonymous-acl-var
    bind :8002
    http-request set-var(txn.req_host) req.hdr(host)
    use_backend vhost if { req.hdr(host) -m str vhost00.example.com }
    [...]
    use_backend vhost if { req.hdr(host) -m str vhost99.example.com }
```

- Way better: **68k req/s**.
- Still hard to maintain.
- We can do better.

Why not a map file?

Using a map

```
# Strip potential port number and convert hostname to lowercase. This
# value is used to select a backend from the backends.map file. Use default
# backend host is not found in the map.
# map file format:
#
#    HOSTNAME  BACKEND_TO_USE
frontend backend-map-default
    bind :8003
    use_backend %[req.hdr(host),word(1,:),lower,map_str(backends.map,default)]
```

- One single line.
- Easier to maintain.
- (Almost) constant lookup time and latency (**150k req/s** and **427µs**, **148k req/s 427µs** with 1000 entries).
- Can handle mixed case and port number.
- Can specify a default entry.

ACL vs map files

```
# Backends.map is:
#   HOSTNAME BACKEND_NAME
# host value is not sanitized for the sake of this example.
acl is_valid_host req.hdr(host),map_str(backends.map) -m found
http-request deny if !is_valid_host
```

```
# -M load the file pointed by -f like a map file:
# pattern is 1st column instead of using the full line as pattern.
acl is_valid_host req.hdr(host) -m str -M -f backends.map
http-request deny if !is_valid_host
```

- ACL can be loaded from a map file if **`map`** return value is not used.
- Results are very close.
- Same file can be used for several purposes.

Map files conclusions

- Easy to maintain.
- Very low lookup time if data is normalized.
- Can be used with any fetch sample like **`src`**, **`req.hdr`** (hostname or client ID), **`req.cook`**, …
- Can be live updated from the runtime API (**`set map <map> [<key>|#<ref>] <value>`**), rules (**`http-request set-map`**) or with *lb-update* module (enterprise edition).
- Can be used to define variables such as thresholds:

```
tcp-request connection set-var(txn.max_conn_allowed)
str(max_conn_allowed),map(settings.map,10)
```

Stick tables

What are stick-tables?

- In-memory efficient key-values tables to store various information.
- Only existing keys consume memory.
- Initial goal: handle server affinity when cookies are not usable (TCP mode).
- Now: can store lots of statistics (connection and request rates, bytes in or out, custom values such as `gpc` or `gpt`).
- Can be replicated to another HAProxy server.
- Can be aggregated using the *Global Profiling Engine* (AKA *stick-table-aggregator*) in Enterprise Edition.

Stick table definition

```
# Dummy backend used to track statistics per client IP. The table is
# configured to compute the connection rate within 1m time frame and
# compute errors within 10m.
backend per-ip
    stick-table type ip size 10k expire 24h store conn_rate(1m)
store http_req_rate(1m) store http_err_rate(10m) store
http_fail_rate(10m)

# Dummy backend used to track if a file has been downloaded in the
# last hour. Instead of storing the file path, we use the base32 CRC
# of the path which is more efficient.
backend per-url
    stick-table type binary len 8 size 10k expire 24h store
http_req_rate(1h)
```

Stick table definition

- Only one table per backend.
- Table named after its backend.
- Need to define *dummy* backends to create more stick-table.
- Are referred by their names in configuration:
  `table_http_req_rate(per-ip)`.
- Can be queried from runtime API:

```
# List all tables
socat unix:haproxy.sock - <<< "show table"

# Show a table content
socat unix:haproxy.sock - <<< "show table per-ip"
```

Stick table definition

All tables are shown in the statistic page.

my-app

		Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server										Extra modules		
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	ssl	h2	h1
	Frontend				0	10	-	10	11	524 274	10			19 782 990	66 636 350	347 060	0	0					OPEN									ssl	h2	h1
☐	demo-server	0	0	-	0	10		0	10	-	10	10	9s	570	830		0		0	0	0	0	*no check*		1/1	Y	-			-	ssl		
	Backend	0	0		0	10		0	10	52 428	10	10	9s	19 782 990	66 636 350	0	0		0	0	0	0	13s UP		1/1	1	0		0	0s		ssl	h2	h1

Choose the action to perform on the checked servers : Apply

per-ip

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server										Extra modules		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	ssl	h2	h1
Backend	0	0		0	0		0	0	1	0	0	?	0	0	0	0		0	0	0	0	13s UP		0/0	0	0		0			ssl	h2	h1

per-url

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server										Extra modules		
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle	ssl	h2	h1
Backend	0	0		0	0		0	0	1	0	0	?	0	0	0	0		0	0	0	0	13s UP		0/0	0	0		0			ssl	h2	h1

Stick table in peers section

```
peers my-app
    # requires at least 1 server
    server "$HAPROXY_LOCALPEER" # this host name

    # table NAME DEFINITION
    # instead of
    # stick-table DEFINITION
    table per-ip type ip size 10k expire 24h store conn_rate(1m)
store http_req_rate(1m) store http_err_rate(10m) store
http_fail_rate(10m)

    table per-url type binary len 8 size 10k expire 24h store
http_req_rate(1h)
```

Stick table in `peers` section

- Can be replicated to remote peer or aggregator (Enterprise only).
- **Latest update wins!**
- Server name must match hostname or peer name (`-L` option).
- No need for extra *dummy* backends (do not pollute statistics page).
- Are referenced by `PEERS_NAME/TABLE_NAME` scheme: `table_http_req_rate(my-app/per-ip)`.

```
# List all tables
socat unix:haproxy.sock - <<< "show table"

# Show a table content
socat unix:haproxy.sock - <<< "show table my-app/per-ip"
```

Dos and don’ts

```
# Track per client IP statistics.
http-request track-sc0 src table per-ip
```

- Track layer 4 data (src, ports) at tcp-request connect
- Track TCP data from payload at tcp-request content

```
# Source IP address is only made at connection time.
# Not for all requests.
tcp-request connect track-sc0 src table per-ip
# Capture ssl version for TCP passthrough proxy
tcp-request inspect delay 1s
tcp-request session track-sc1 req.ssl_ver
```

Conclusions

- Use peers section if multiple tables are required.
- Track values as soon as possible to prevent multiple lookups per connection.
- Tables are useful for rate limiting and abuse detection.
- Table content is lost on reload except when using a `peers` section.

Miscellaneous optimizations

ACL definition

```
acl allowed_host src 10.0.0.10 10.0.0.20 10.0.0.30
acl allowed_host src -m ip -f office-fr.acl
acl allowed_host src -m ip -f office-hr.acl
acl allowed_host src -m ip -f office-us.acl
acl allowed_host src -m ip -f office-ca.acl
```

```
# Better solution
acl allowed_host src -f office-fr.acl -f office-hr.acl -f
office-us.acl -f office-ca.acl 10.0.0.10 10.0.0.20 10.0.0.30
```

- Items are OR'd
- Inefficient: creates 1 lookup tree per line. Lookup forest!
- **132k req/s** vs **134k req/s**. Gap increases with statements.

ACL

- Multiple definitions can be used if criterions are different:

```
acl has_client_id url_param(client-id) -m found
acl has_client_id req.hdr(x-client-id) -m found
```

- Order matters.
- On-demand evaluation.
- Evaluation is stopped as soon as possible:

```
# has_client_id is not evaluated if allowed_host is true
http-request deny if !allowed_host !has_client_id
```

x-forwarded-for

```
# (1) Adds xff. Potential security issue
http-request add-header x-forwarded-for %[src]
# (2) Removes and adds xff
http-request del-header x-forwarded-for
http-request add-header x-forwarded-for %[src]
# (3) Replaces xff
http-request set-header x-forwarded-for %[src]
```

```
# (4) Adds xff. Potential security issue
option forwardfor
# (5) Replaces xff
option forwardfor
http-request del-header x-forwarded-for
```

X-forwarded-for (results)

1	`add-header`	**135k req/s**
2	`del-header` + `add-header`	**132k req/s**
3	`set-header`	**133k req/s**
4	`option forwardfor`	**152k req/s**
5	`option forwardfor` + `del-header`	**150k req/s**

- Use **`option forwardfor`**.
- Additional **`del-header`** rule can be used if security is a concern.
- Use **`tcpdump`** to validate results sent to the server.

x-forwarded-proto

```
http-request set-header X-Forwarded-Proto https if { ssl_fc }
http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
```

```
# use iif converter which is cheaper than 2 http-request rules
http-request set-header X-Forwarded-Proto %[ssl_fc,iif(https,http)]
```

- With 2 rules: **144k req/s**
- Using `iif`: **146k req/s**

HTTP and HTTPS

```
frontend http
    bind *:80
    # full ruleset
    use_backend my-backend

frontend https
    bind *:443 ssl crt /etc/haproxy/certs
    # full ruleset
    use_backend my-backend
```

HTTP and HTTPS (2)

- Frontends can use several binds
- Even with different methods (IPv4, IPv6, abns, unix socket, …)
- Easier to maintain

```
frontend web
    bind *:80
    bind *:443 ssl crt /etc/haproxy/certs
    # full ruleset
    use_backend my-backend
    # Let the client remember to use https
    http-after-response set-header Strict-Transport-Security
"max-age=31536000; includeSubDomains; preload;" if { ssl_fc }
```

HTTP and HTTPS (3)

```
# (1) http -> https
redirect scheme https code 301 if !{ ssl_fc }
```

```
# (2) http-request redirect is better than redirect
http-request redirect scheme https code 301 if !{ ssl_fc }
# (3) If not using standard ports. Does it really make sense?
http-request redirect code 301 location
https://%[req.hdr(host),word(1,:)]:8443%[pathq] if !{ ssl_fc }
```

- 1: **`redirect scheme`** is done after **`http-request`** rules. (**118k req/s**).
- 2: Careful if not using standard ports! (**156k req/s**).
- 3: Does this example really makes sense? (**145k req/s**).

Rule of thumb **`http-request redirect`** from http to https should be the first rule.

`last_rule_file` and `last_rule_line`

- Useful to determine latest **`http-request`** or **`tcp-request`** rule in request processing.
- Add these values to a log line:

```
# Add last rule information to standard log line.
# HTTP_LOG should be declared in the global section using a
# setenv directive.
log-format "${HTTP_LOG} lr:%[last_rule_file]:%[last_rule_line]"
```

```
Nov  2 17:08:27 lab-inj02 haproxy[24499]: 198.18.0.101:47082
[02/Nov/2022:17:08:27.161] xforwardedfor-option-del demo-server/<NOSRV>
0/-1/-1/-1/0 200 74 - - LR-- 1/1/0/0/0 0/0 "GET / HTTP/1.1" lr:haproxy.cfg:97
```

Conditional configuration

```
peers my-app
    # The local peer has to be defined with no option
    server "$HAPROXY_LOCALPEER"

    # Add some replicas if required. A peer name MUST match the server
    # hostname or the name given with the -L option to HAProxy.
    .if defined(IS_CLUSTER)
        bind *:10000
        # If current peer is lb1, we need to add lb2
        .if streq("$HAPROXY_LOCALPEER","lb1")
            server lb2 192.168.255.2:10000
        .endif
        # Same in the other way round.
        .if streq("$HAPROXY_LOCALPEER","lb2")
            server lb1 192.168.255.1:10000
        .endif
    .endif
    # Tables definition goes here
```

String escaping

- Use weak (`"`) or strong quoting (`'`) instead of backslash (`\`) space escaping.
- Easier to read and modify.
- Weak quotes: allow variable expansion.
- Strong quotes: nothing is interpreted. Use it with regular expressions.

```
http-request set-header x-comment this\ is difficult\ to\ read
http-request set-header x-comment "this is easier to read"
# Protect regexp strings and arguments (Check official documentation)
http-request set-path '%[path,regsub("^/(here)(/|$)","my/\1",g)]'
```

Who spotted the bug in first rule?
Missing `\` between `is` and `difficult`.

Thank for your attention.

Any questions?