

Observability tips for HAProxy

Willy Tarreau

(willy@haproxy.org)

Dotscale 2018

Definition

Observability:

"control theory, observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs" (*wikipedia*)

In short for us: *WTF is going on ?*

Observability vs monitoring

- Monitoring tells you *how well* something works (or not)
- Observability helps you *detect* what is not working and *why*

=> *You monitor an observable system.*

Observability is important

- can't rely on impatient users' complaints anymore
=> *detect and fix trouble **not reported yet***
- stop adding duct tape, address **root causes!**
- improve users experience where it **matters**

The LB as an observation tower

- **central** place
- in distributed systems, one **LB per level**, even better when sidecar
- sees multiple targets, eases **comparisons**
=> *draw references*
- **Trusted** low level component & excellent transparency
- many LB **decisions** actually depend on **metrics** related to performance and observability
- already logs, provides long-term **references**

Why not look from other points ?

- you should! Especially in **distributed** systems (*microservices, etc*)!
- but often it's **too late** when the first incident happens!
- with **existing** LB's logs, it's already possible to do a lot

Note: see OpenTracing and Prometheus

What does the LB see ?

- global **failures** (aborts, timeouts)
- abnormal **delays** caused by network retransmits
- connection failures and **retries** caused by bad tuning (*eg: conntrack*)
- connection **slowdowns** caused by inefficient firewall policies (*#rules*)
- ...

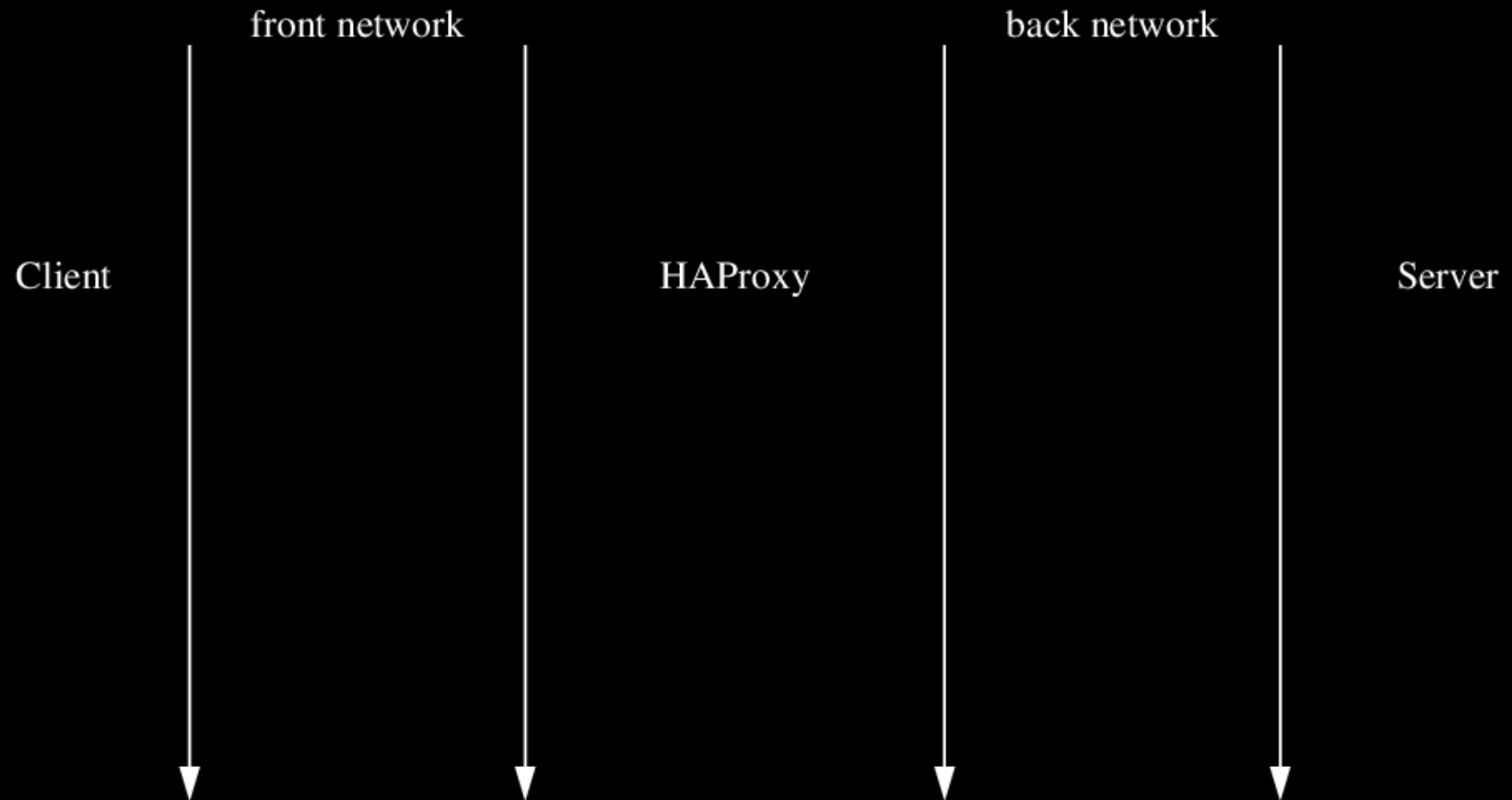
What does the LB see (...) ?

- **client-side** issues (*BW limitations*)
- **per-URL** processing time (*application issues, svc partners*)
- per-node vs **per-cluster** variations
=> *narrow down to individual node or shared resource*
- **deployment** issues : new occasional error on a specific page, can be addressed before going full-scale

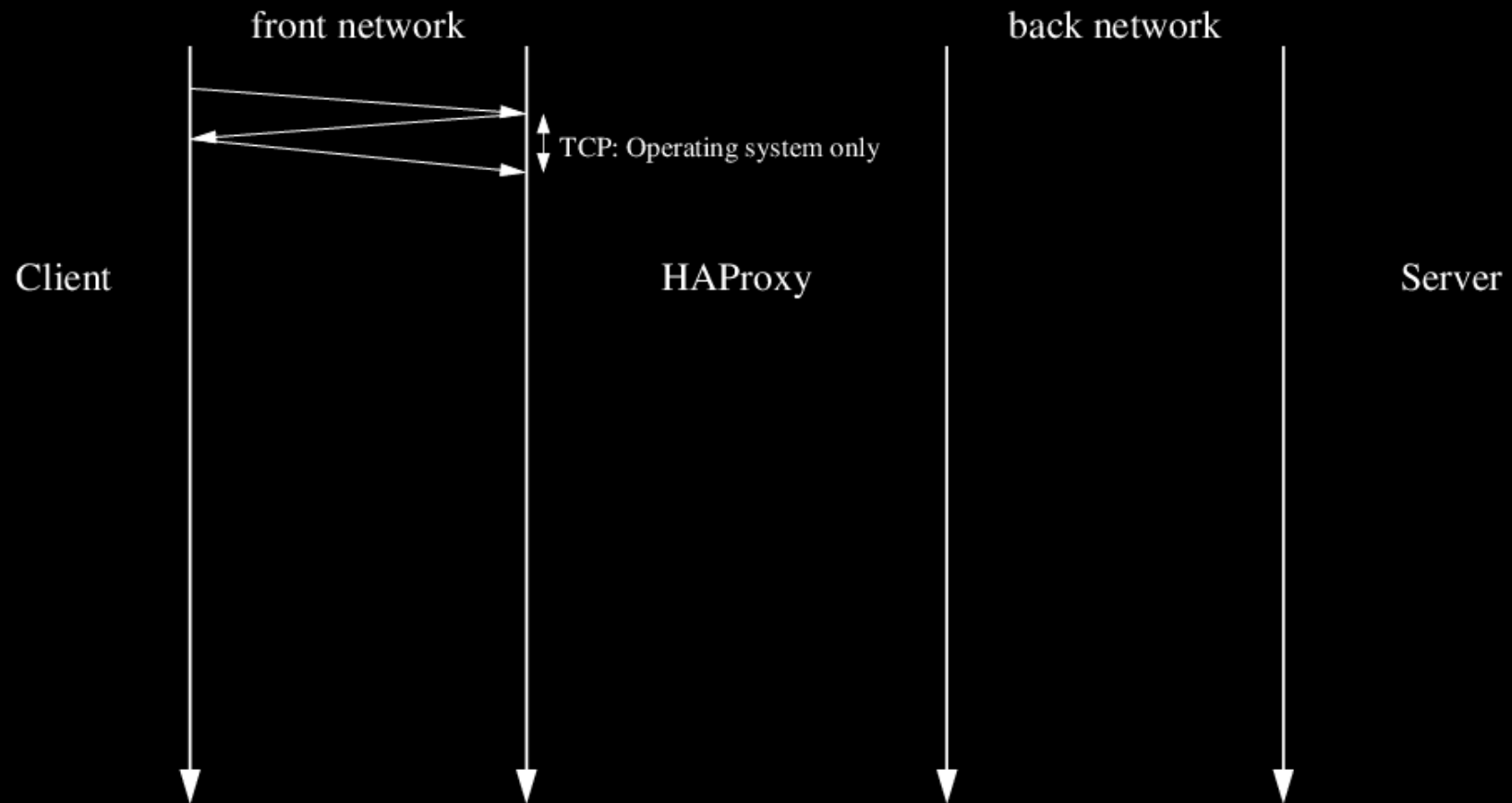
Accessing metrics in HAProxy

- Logs :
 - Hlog, ELK, Prometheus, ...
 - Provides **unique-id** for tracing/event correlation
- Stats :
 - Stats page, CLI, hatop
- Stick-tables (per arbitrary key like IP, URL, cookie) :
 - Byte count, cumulated/concurrent conns, errors, ...

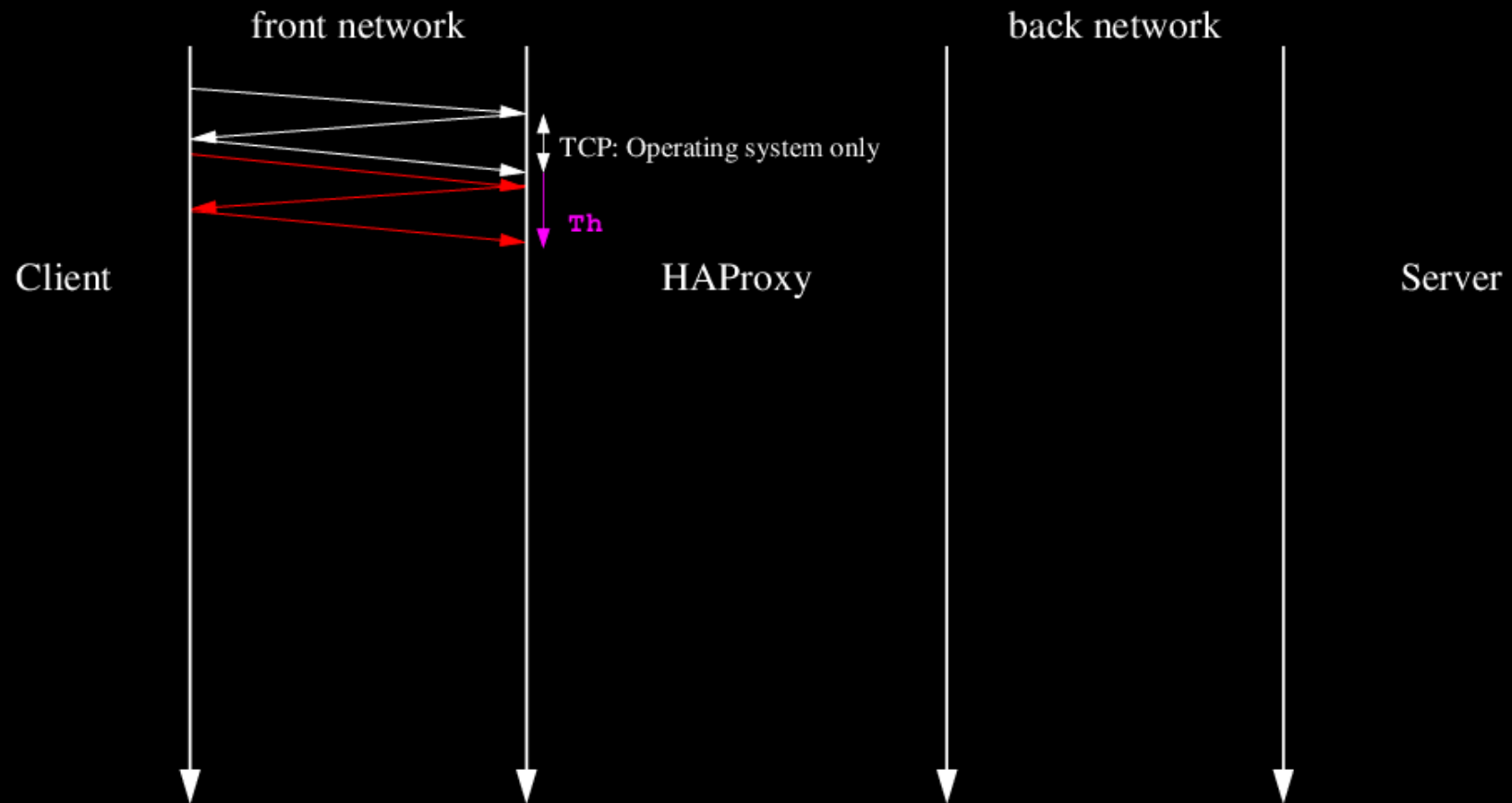
Sequence of events on HAProxy



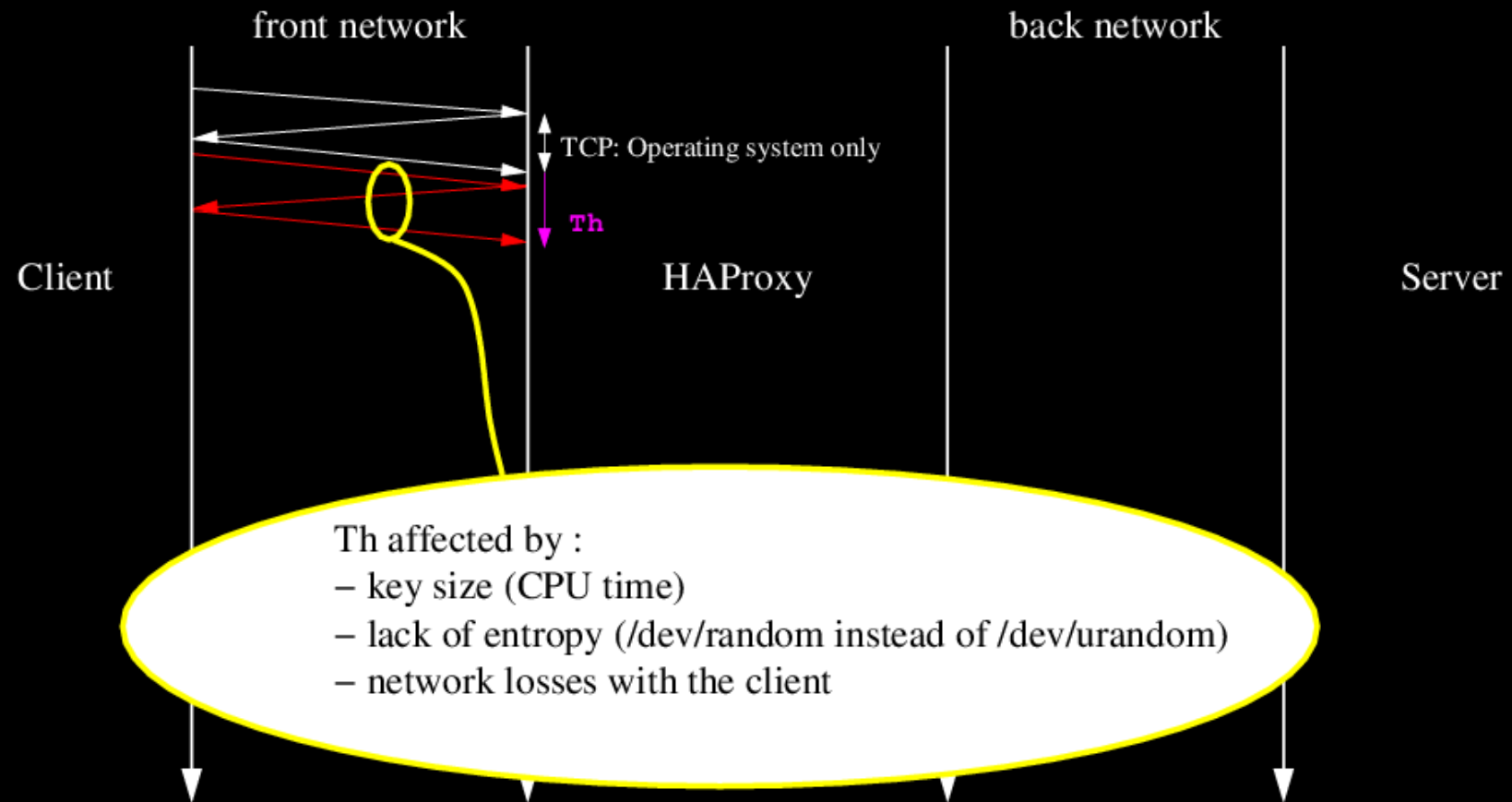
Sequence of events on HAProxy



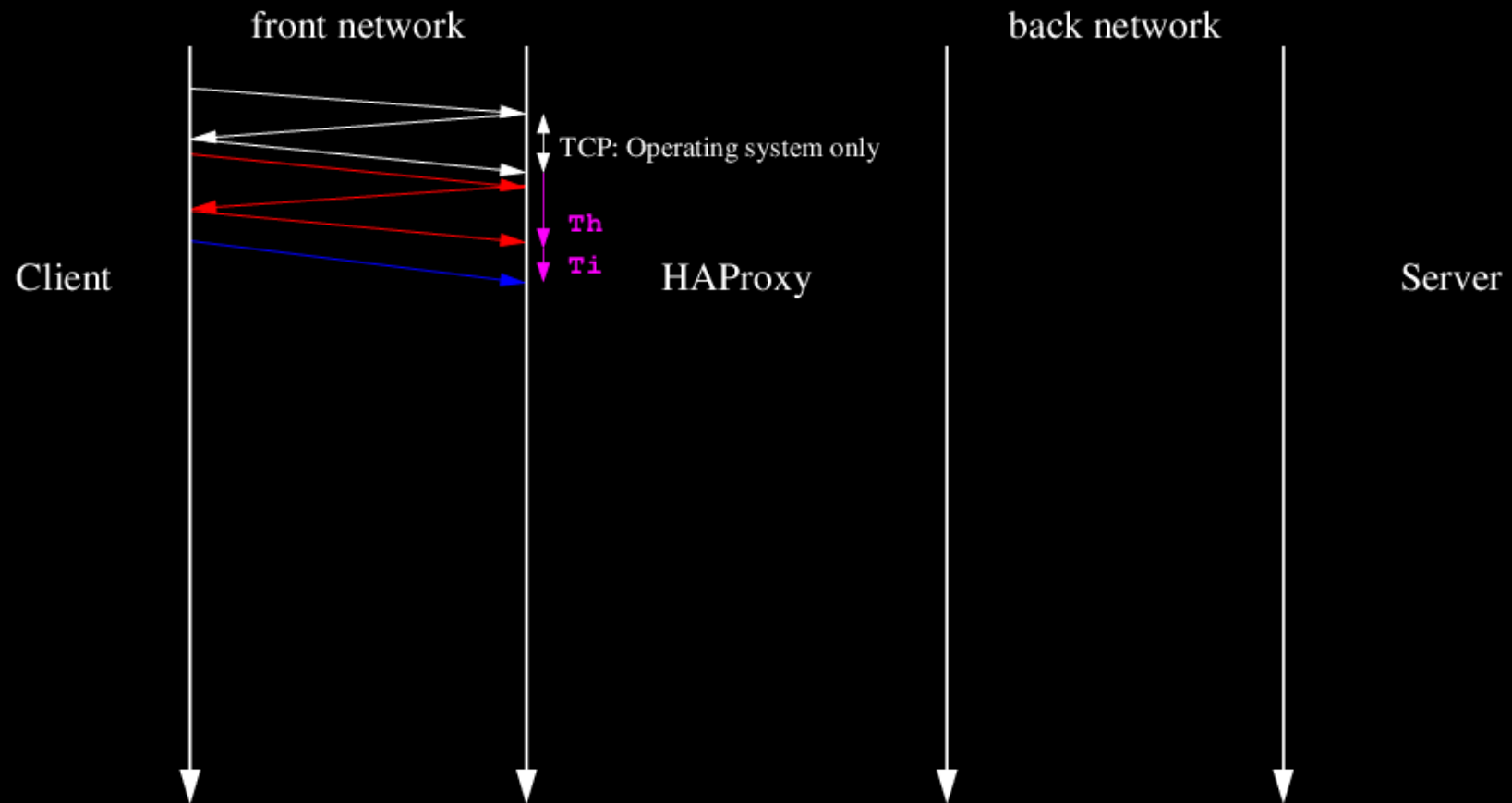
Sequence of events on HAProxy



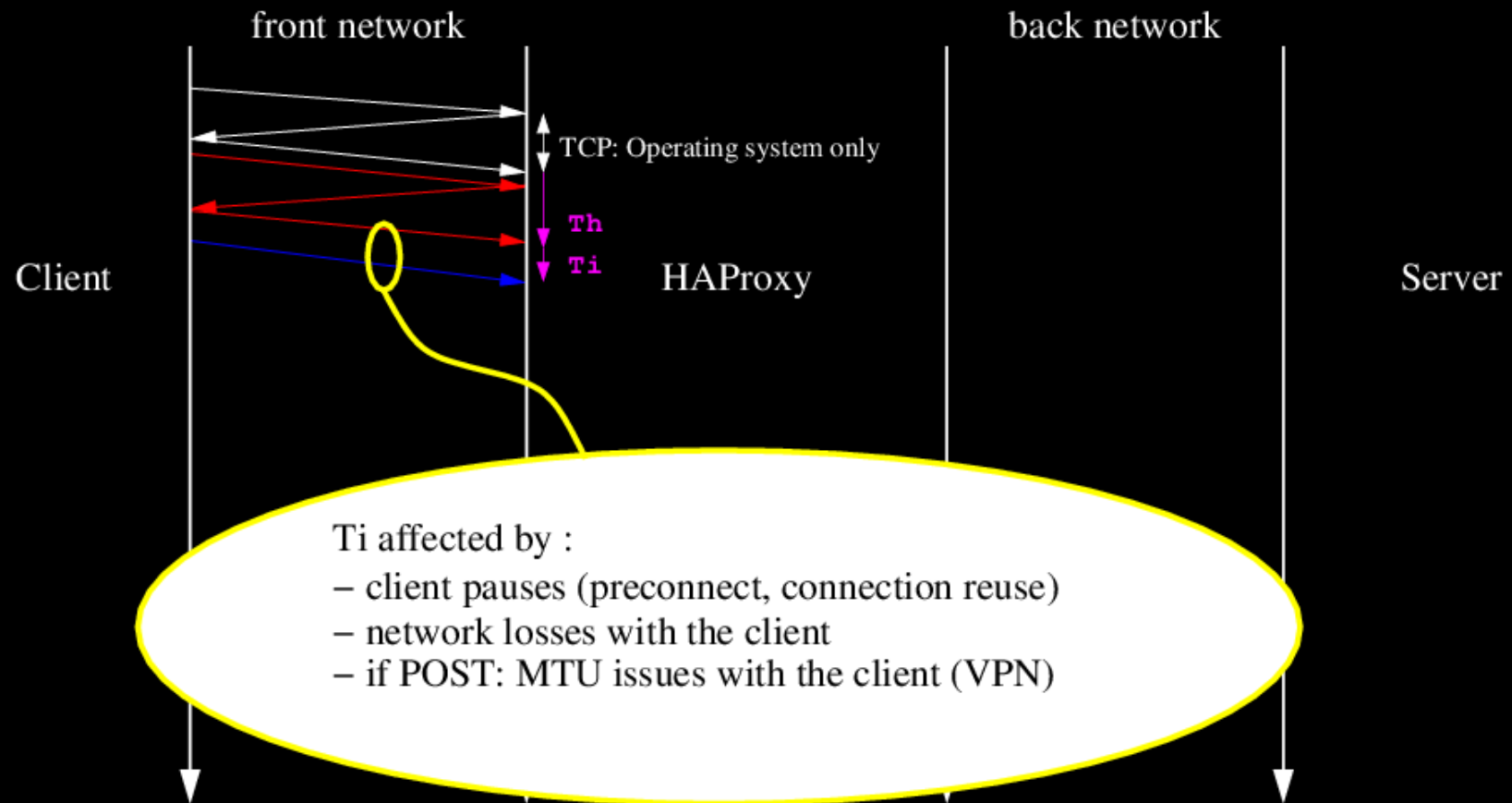
Sequence of events on HAProxy



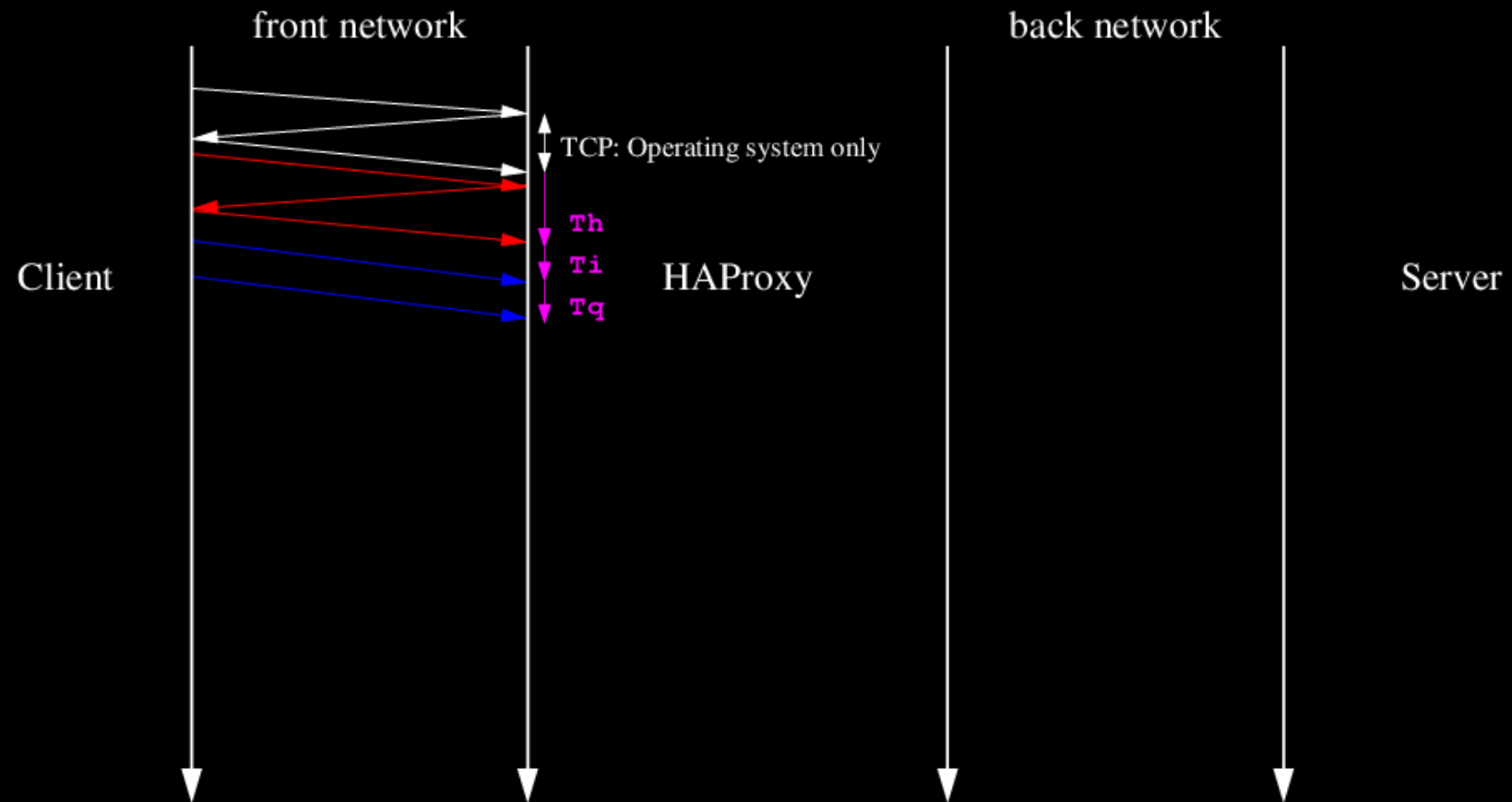
Sequence of events on HAProxy



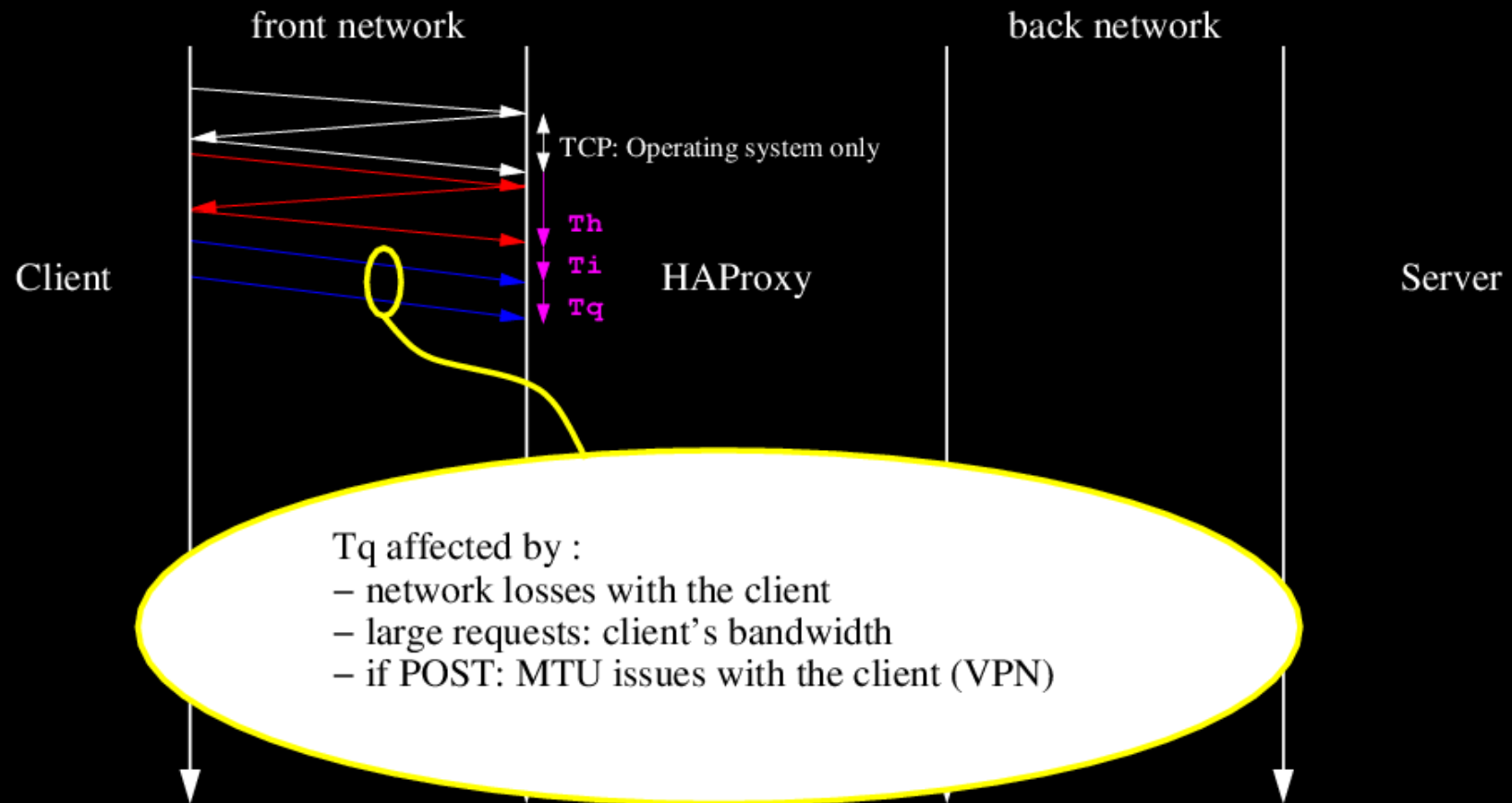
Sequence of events on HAProxy



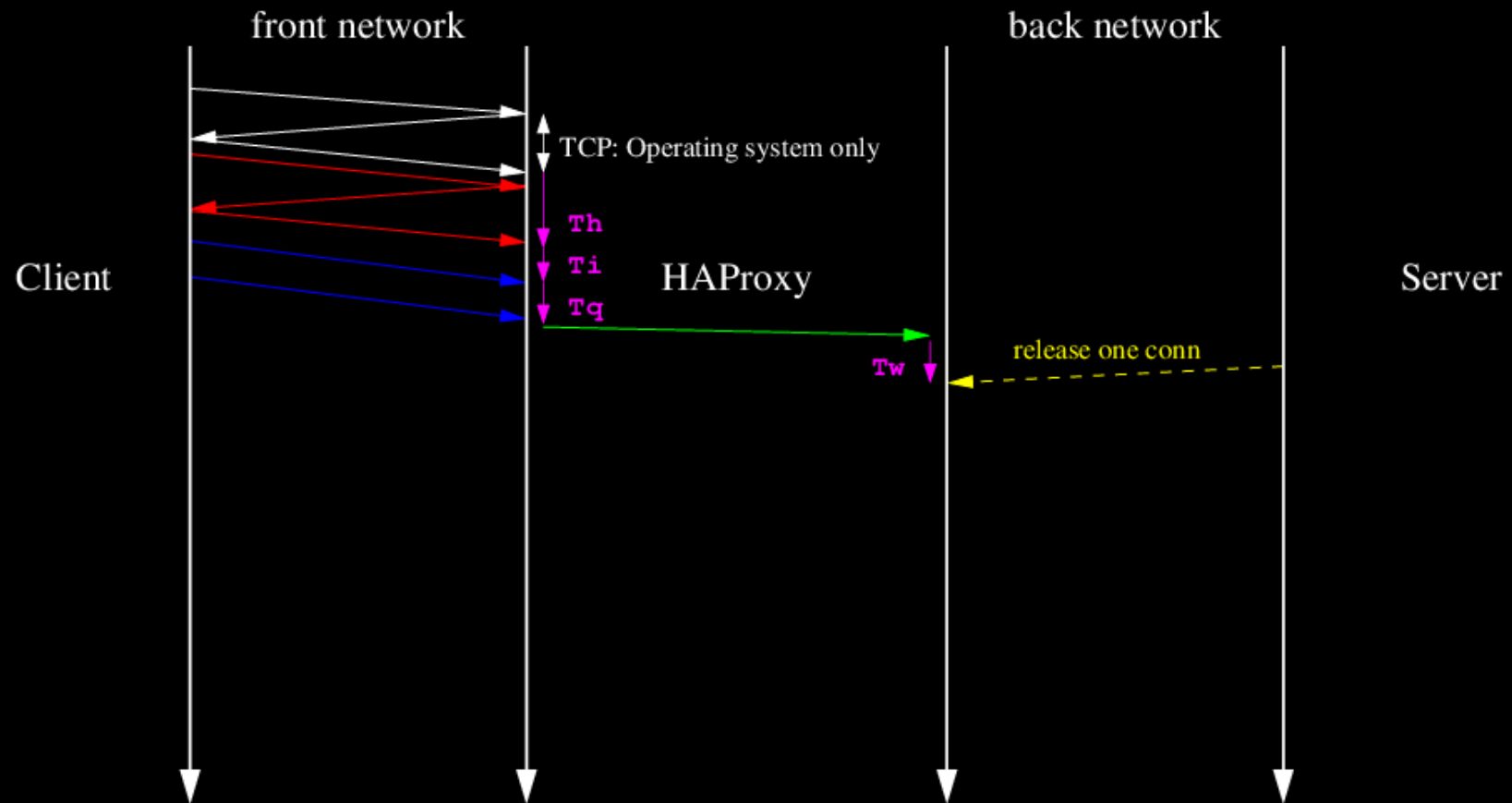
Sequence of events on HAProxy



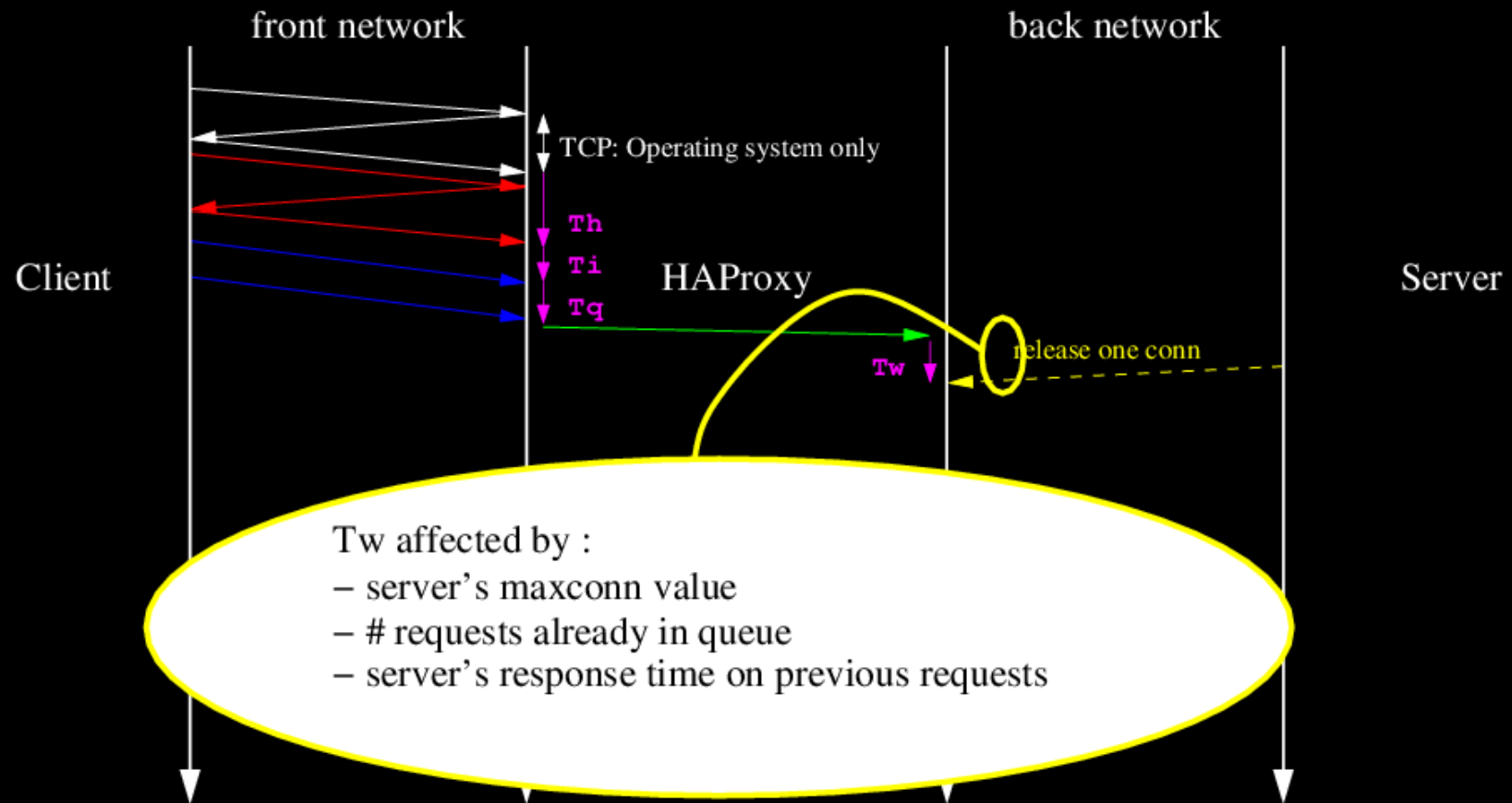
Sequence of events on HAProxy



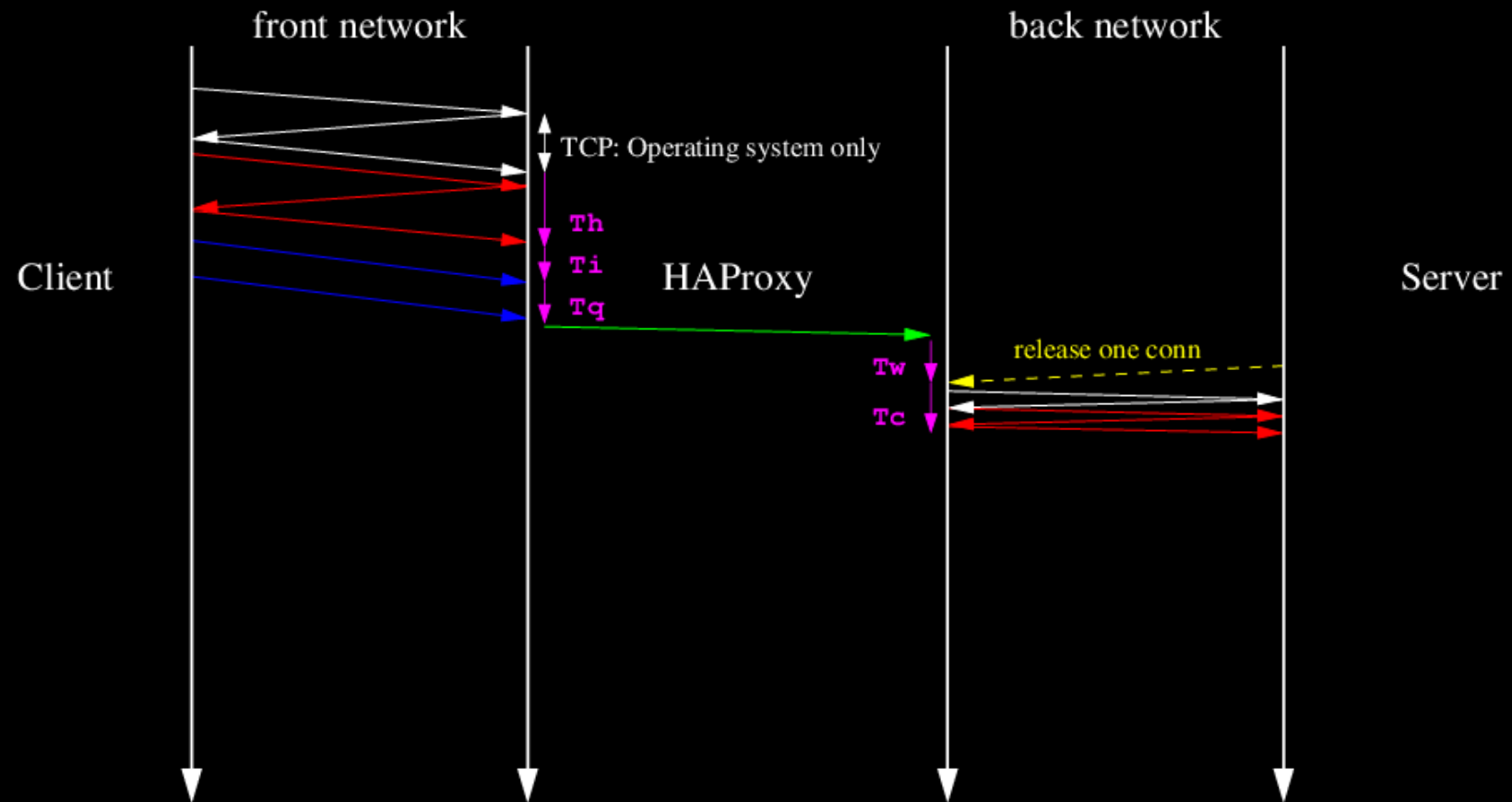
Sequence of events on HAProxy



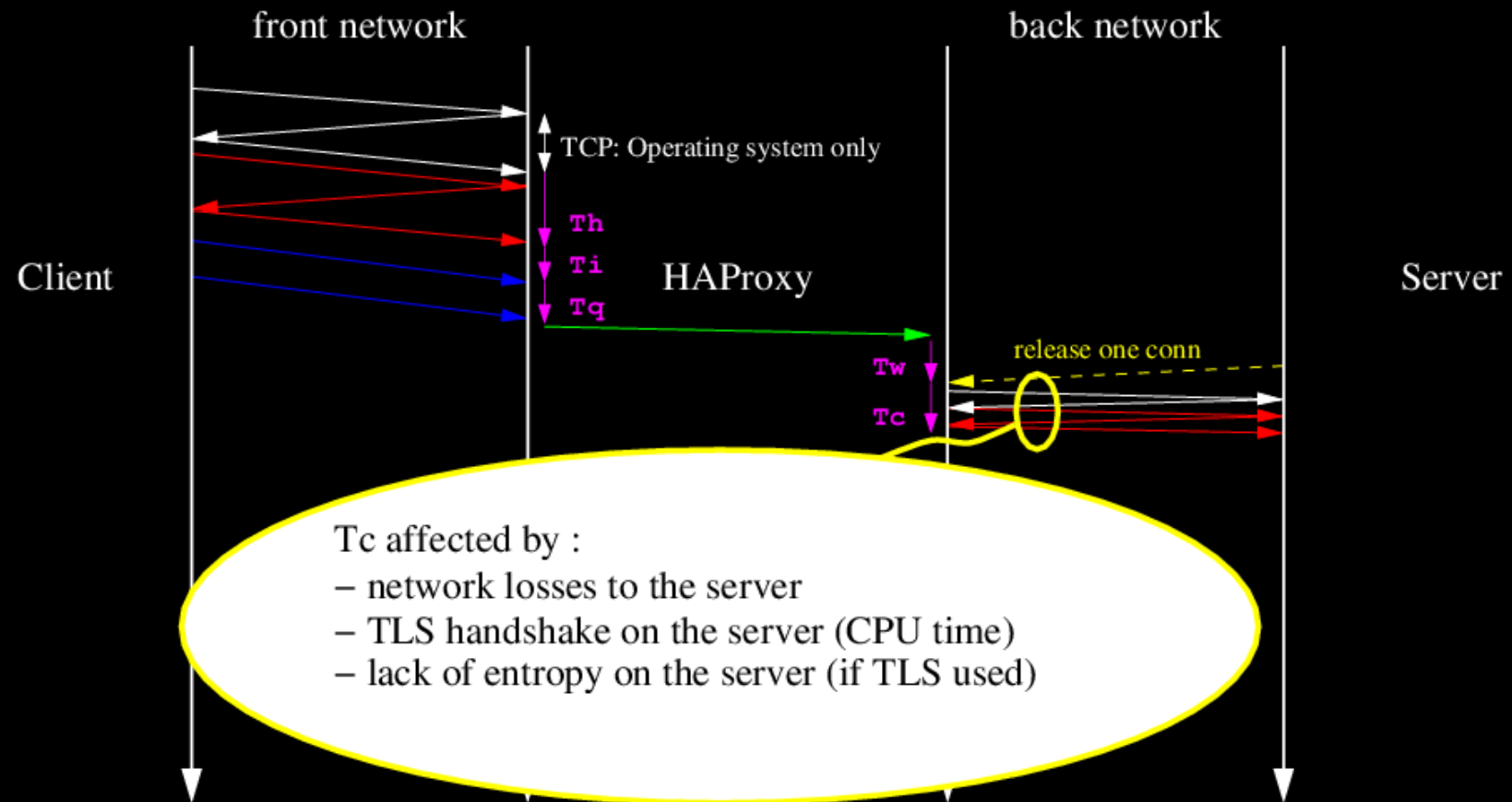
Sequence of events on HAProxy



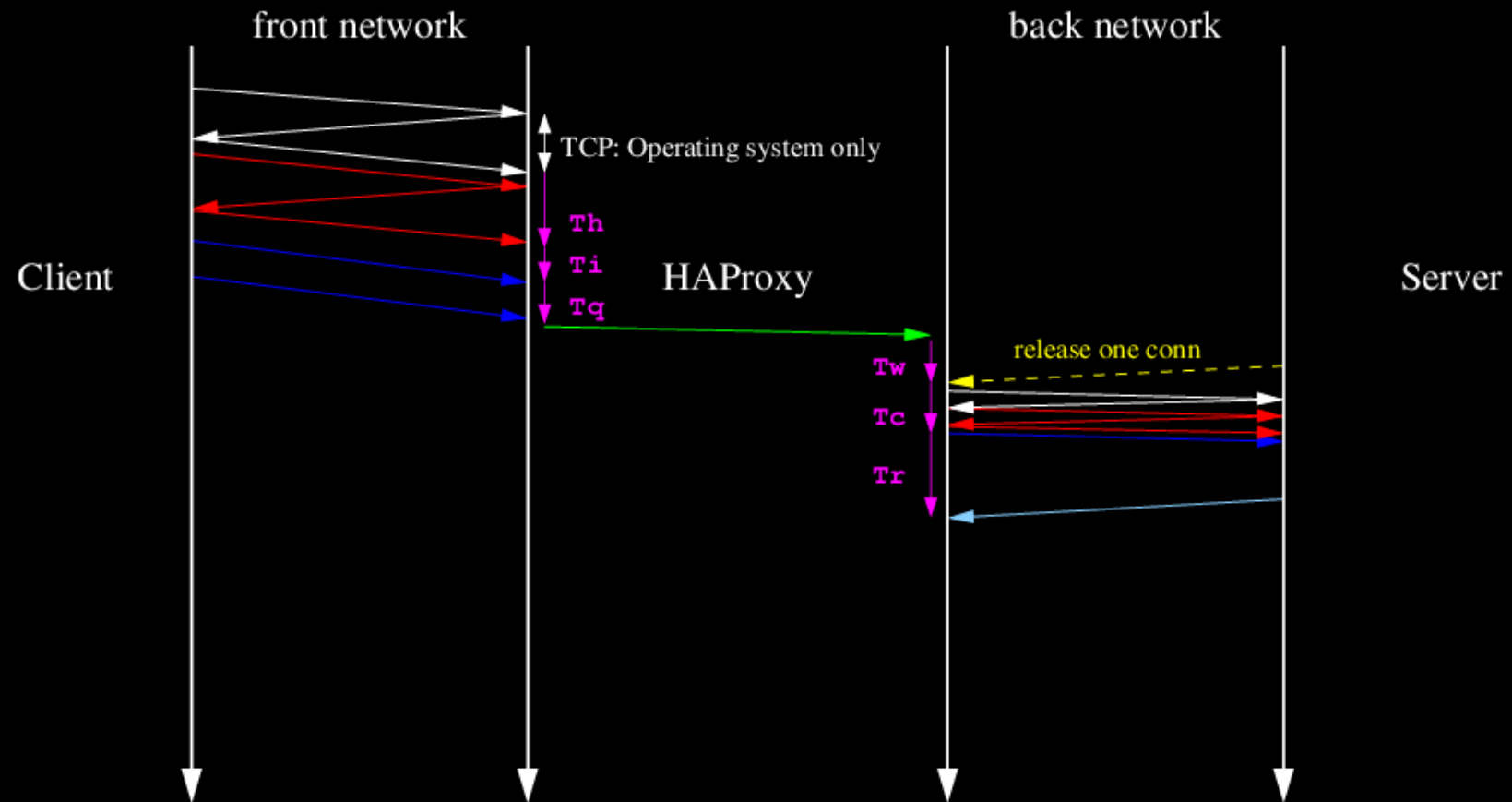
Sequence of events on HAProxy



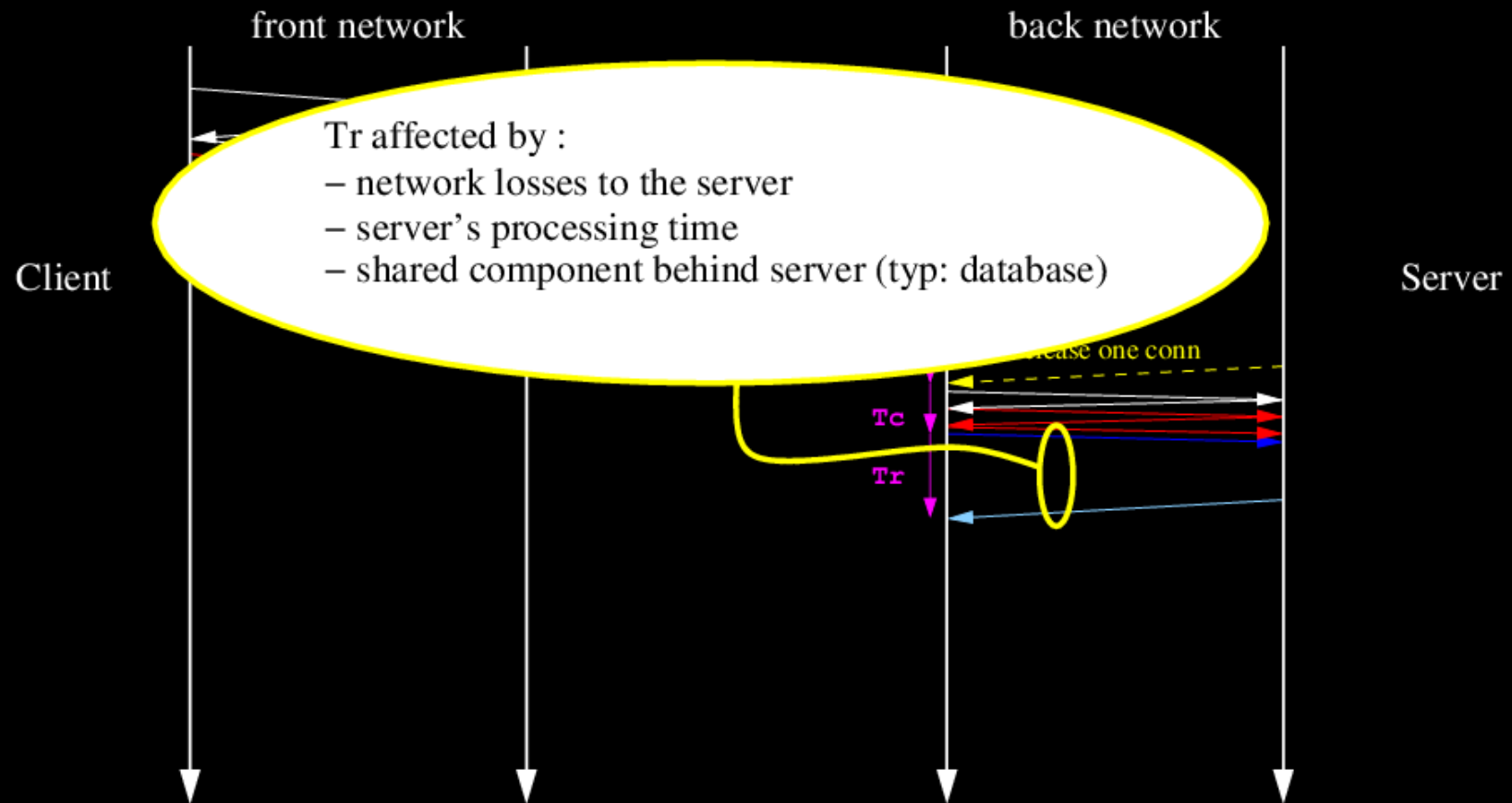
Sequence of events on HAProxy



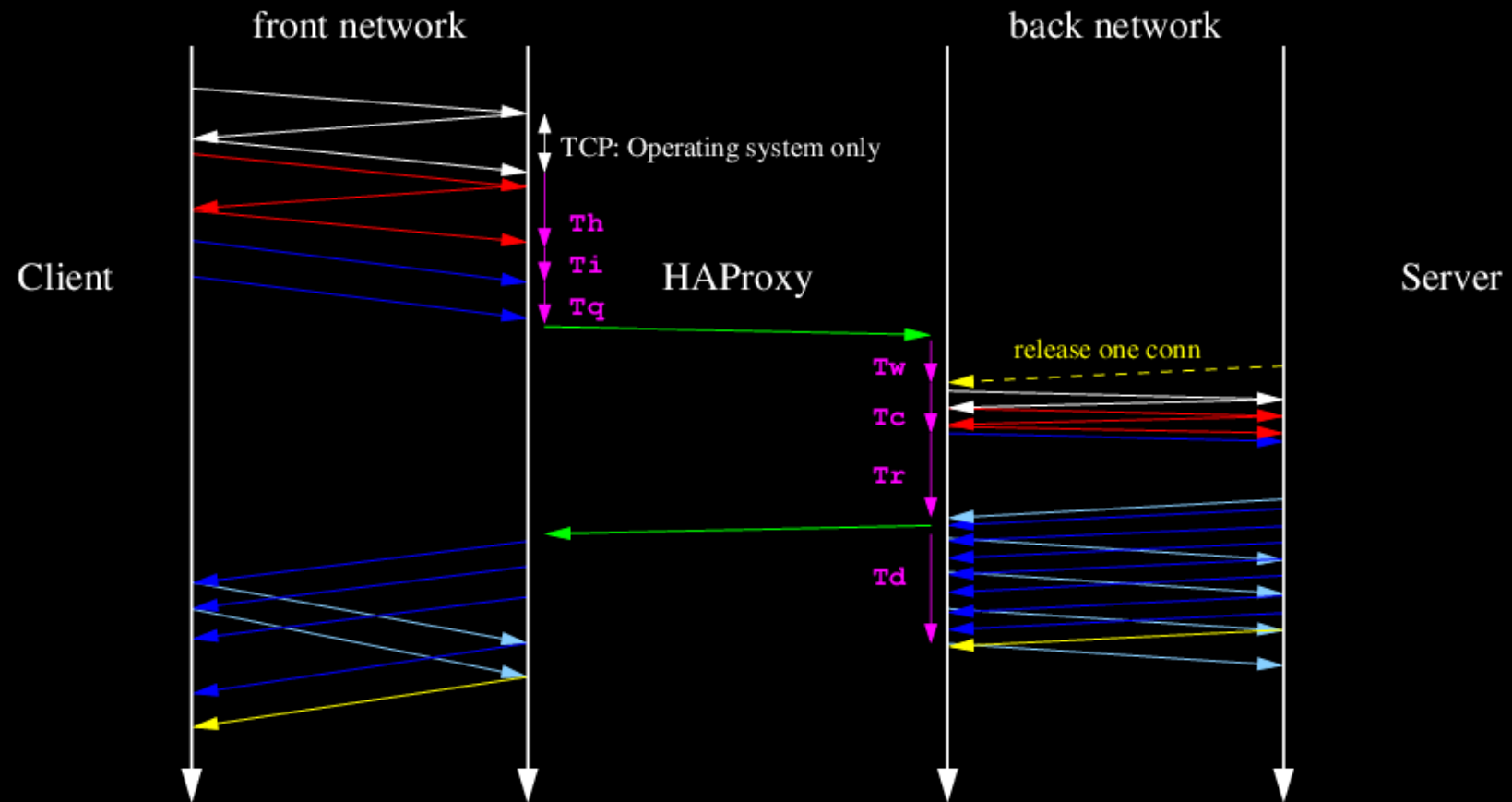
Sequence of events on HAProxy



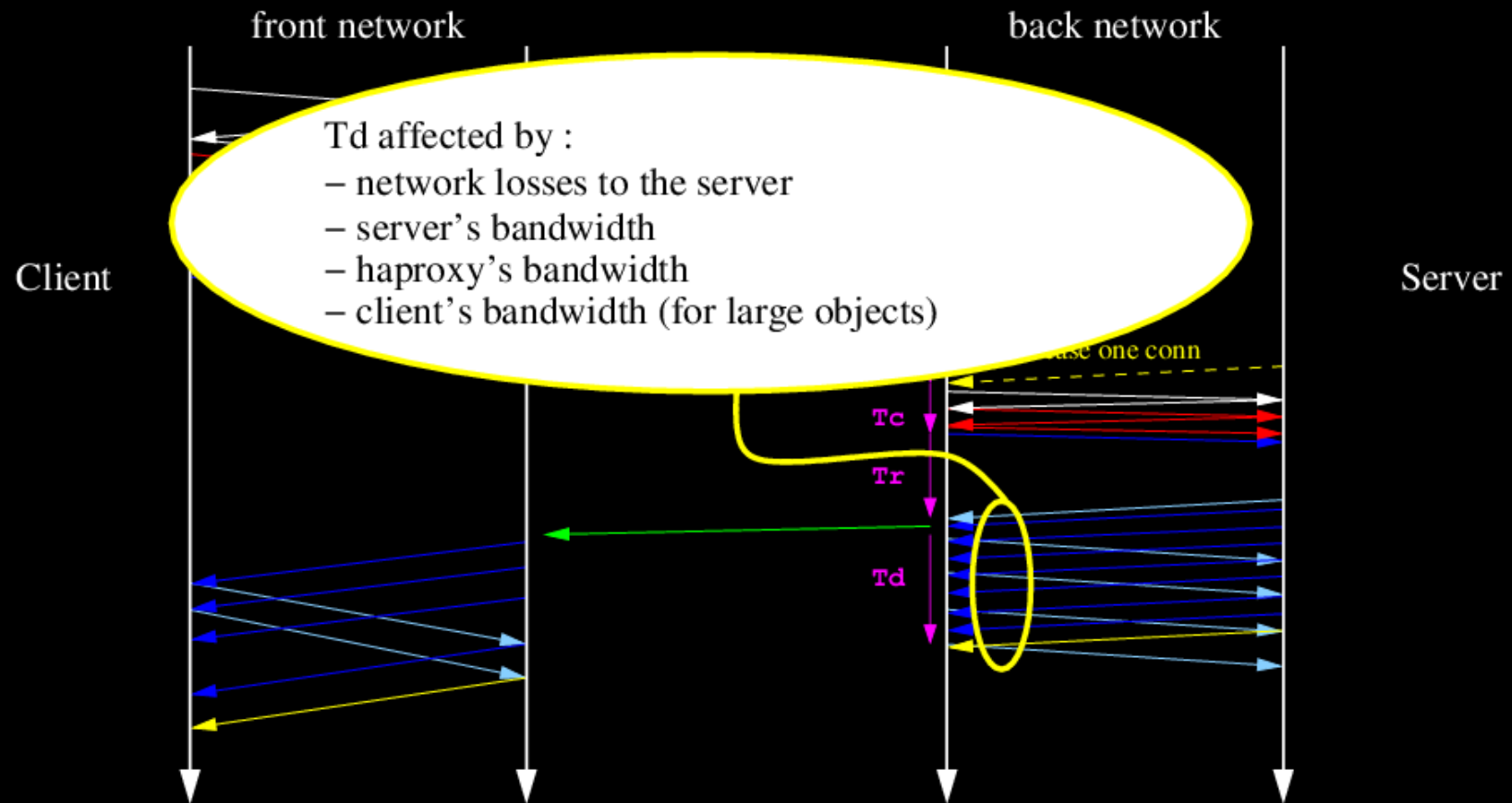
Sequence of events on HAProxy



Sequence of events on HAProxy



Sequence of events on HAProxy



More timers to come in HAProxy 1.9

- HAProxy now supports heavier per-request workloads (Lua, device identification, ...)
- Processing times over **200 µs** can become noticeable

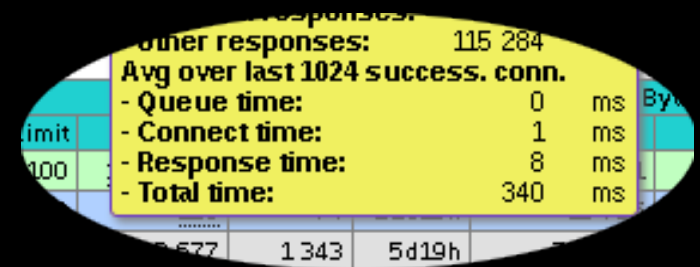
Actions:

- log per-request **total CPU time** spent in **analysers**
- log per-request **total CPU time** spent in **TLS handshake**
- log per-request **total latency** added by other tasks
- Ability to **kill** offending tasks
- Ability to **alert** on high latencies

=> make HAProxy as observable as other components

Event timing reports

- Timers are averaged in the stats
 - Each timer appears in the logs
 - **Hallog -rt/-RT/-pct** for quick analysis
 - Each timer crossing a limit triggers a **timeout**
 - Each abort at a specific step causes a **hard error**
- => termination codes



other responses:		115 284
Avg over last 1024 success. conn.		
- Queue time:	0	ms
- Connect time:	1	ms
- Response time:	8	ms
- Total time:	340	ms

```
haproxy[14389]: 10.0.1.2:33317 [06/Feb/2018:12:14:14.655] http-in  
static/srv1 10/0/30/69/109 200 2750 - - SDNN 1/1/1/1/0 0/0 {haproxy.org}  
{ "GET /index.html HTTP/1.1"
```

Timers

Term code

Cookie code

Termination codes

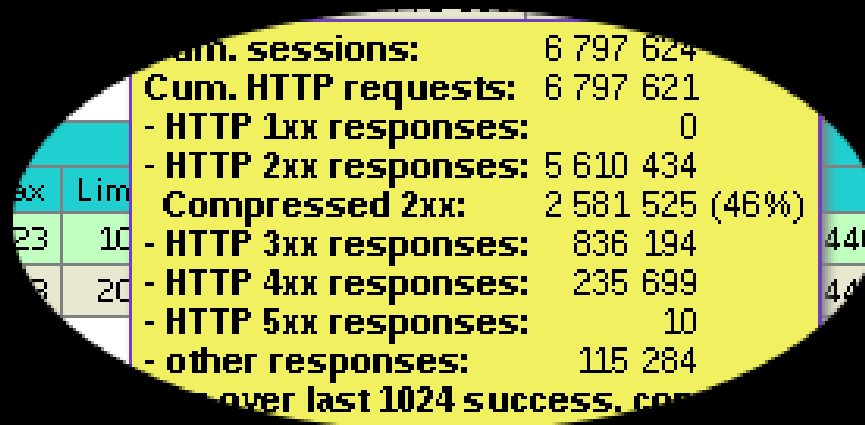
- Distinguish between **timeout** and **abort**
- Indicate **whom** (*client, server, haproxy, kill, ...*)
- Indicate **when** (*req,queue,connect,response...*)
- Completed by persistence cookie indications
- Filtered and sorted by halog :

```
# halog -tcn|-TCN ... # for filtering
# halog -tc           # for sorting
```

Other relevant metrics : HTTP status distribution

- Stats page: distribution per frontend/backend/server
- Filter by ranges: **halog -hs/-HS**
- Sorted output: **halog -st**

=> graph the **distribution** and watch for **variations** between application deployments



The screenshot shows a terminal window with a yellow background. It displays HTTP status distribution statistics. On the left, there is a table with columns 'ex' and 'Lim'. The main part of the terminal shows a list of statistics with their corresponding counts and percentages.

ex	Lim
23	10
8	20

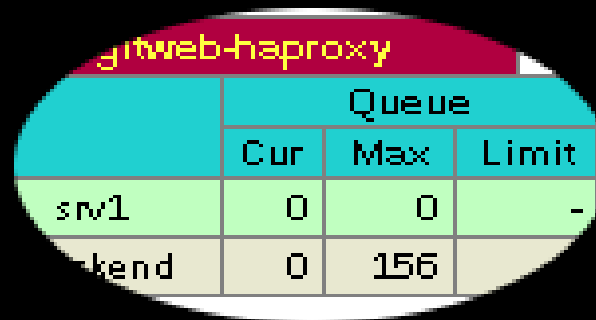
Cum. sessions:	6 797 624
Cum. HTTP requests:	6 797 621
- HTTP 1xx responses:	0
- HTTP 2xx responses:	5 610 434
Compressed 2xx:	2 581 525 (46%)
- HTTP 3xx responses:	836 194
- HTTP 4xx responses:	235 699
- HTTP 5xx responses:	10
- other responses:	115 284

over last 1024 success. con

Other relevant metrics : queue length

- Uses server **maxconn**
- Grows **exponentially** with slowdowns : easy to detect!
- Tells you **how many** extra servers you need
- Reported by **halog -Q/-QS**
- Shown in real time on the stats page per backend/srv

=> If you watch only one metric, watch this one!



	Queue		
	Cur	Max	Limit
srv1	0	0	-
backend	0	156	

Other relevant metrics : LB fairness

LB algorithm implies **fairness** between servers :

- Equal request count with **roundrobin**
=> **Higher** than average **concurrency** indicates abnormally slow server
- Equal load with **leastconn**
=> **Low req count** indicates abnormally slow server

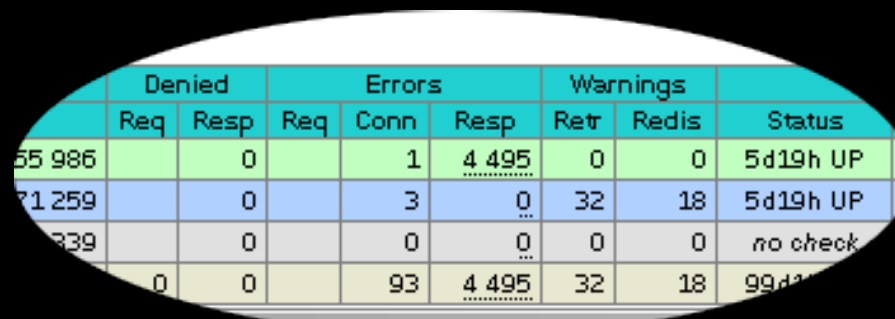
=> *graph relevant values within the farm*

	Queue			Session rate			Sessions				
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot
web01	0	0	-	26	385		59	508	-	5 701 330	5 701 041
web02	0	0	-	26	385		60	421	-	5 690 883	5 690 595
web03	0	0	-	26	385		58	451	-	5 701 198	5 700 934
Backend	0	0		80	1 155		187	1 083	2 050	17 097 566	17 092 570

Other relevant metrics : error rate

- Global: **hlog -e**
- Per server: **hlog -srv**
- Per client IP: **hlog -e -ic** (*detect bad CDN nodes*)
- Per URL: **hlog -ue**
- Stats page: **per frontend/backend/server**
- Stick-tables: **per arbitrary key** using `http_err_rate()`

=> *no threshold, watch for variations*

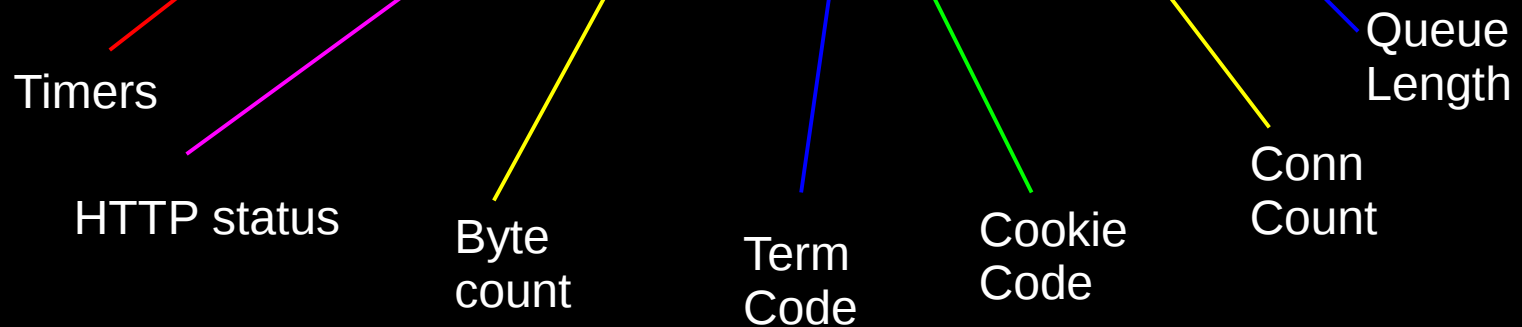


	Denied		Errors			Warnings		Status
	Req	Resp	Req	Conn	Resp	Retr	Redis	
55 986		0		1	4 495	0	0	5d19h UP
71 259		0		3	0	32	18	5d19h UP
339		0		0	0	0	0	no check
	0	0		93	4 495	32	18	99d1h

Useful entries in log-format

- Default httplog format is quite rich
- Can be improved using the **log-format** directive
- Hint: log stick-table stats for similar keys

```
haproxy[14389]: 10.0.1.2:33317 [06/Feb/2018:12:14:14.655] http-in  
static/srv1 10/0/30/69/109 200 2750 - - SDNN 1/1/1/1/0 0/0 {haproxy.org}  
{ } "GET /index.html HTTP/1.1"
```



Tips: sampling : why / when

"I can't enable logs, I have too much traffic!"

- an average syslog server can store **20k** events/s without sweating
- that's 1.7B events/day or **350GB of uncompressed** haproxy logs/day
- compresses to **1TB/month**
- for **\$100** you can store **4 months** with no loss
- have more traffic / not interested in this level of detail ?

```
# log only 5% of requests  
http-request set-log-level silent unless { rand(100) -lt 5 }
```

Tips: selective logging: why / when

- you only want to catch **suspicious** events
- disable logging unless Tc/Tq/Tr/Tw/... is above a certain threshold
- on the fly for **selected keys** from the CLI + **stick-table**
- also see "**option dontlognormal**"
- **WARNING:** you'll lose any valid **reference**

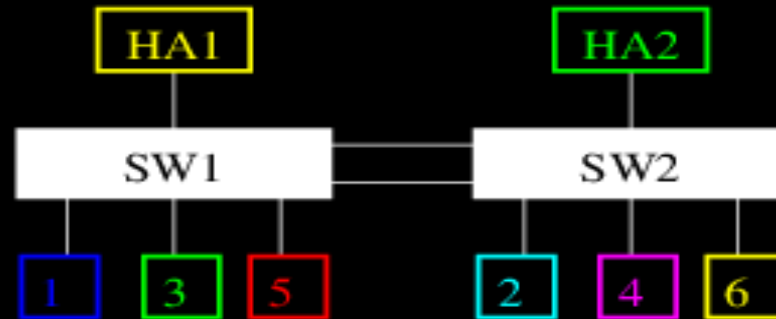
Tips: other halog goodies

- Poorly documented, use **halog --help**
- response time per url: **halog -uat**
- errors per server: **halog -srv**
- Percentiles on req/queue/conn/resp times: **halog -pct**
- detect stolen CPU / swap : **halog -ac ... -ad ...**
- very fast (1-2 GB per second)

=> Use it in production to figure the relevant metrics

Success stories

Customer spotting a broken fiber between two core switches



- Tc from **HA1** to srv 1,2,3,5 always low, srv **4,6** high at 99 pct
- Tc from **HA2** to srv 1,2,4,6 always low, srv **3,5** high at 99 pct
=> **both haproxy and servers out of cause**
- issue rate **stable** at various traffic levels => **not congestion**
- inter-switch link apparently at cause but not for all flows
- inter-switch link made of two fibers balanced on MAC tuple
- thanks to long-term logs, origin could even be identified

Success stories

Customer figuring a wrong web server configuration using /dev/random

- Tc abnormally **high** with lots of random values to several seconds, and only for **TLS**
- timer also covers TLS **handshake**
=> not a network, hardware or performance issue, only server config.
*=> system was regularly running **out of entropy** due to mistakenly using /dev/random as a random source for SSL*

Conclusion

- exploit your stats
- enable logs on LBs, no excuse for not doing it!
- process them automatically, **manually once in a while**
- **compare numbers** between similar objects
- detect anomalies
- fix problems before they are witnessed
- profit :-)

Interesting lectures

- <https://medium.com/@copyconstruct/monitoring-and-observability-8417d1952e1c>
- <https://www.vividcortex.com/blog/monitoring-isnt-observability>
- <http://opentracing.io/documentation/>
- <https://prometheus.io/>