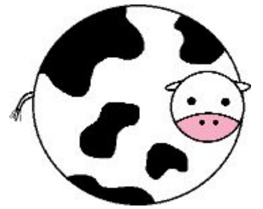


Do You Want to Retry?

Anton Marchukov

About Me

- **Software Engineer** at **Red Hat**.
- **oVirt Community Infra** team.
- **CI** and related **infrastructure**.
- Lot of **automation** in **Python**.
- **DevOps** advocate.

oVirt is free, open-source virtualization management platform based on the KVM hypervisor.

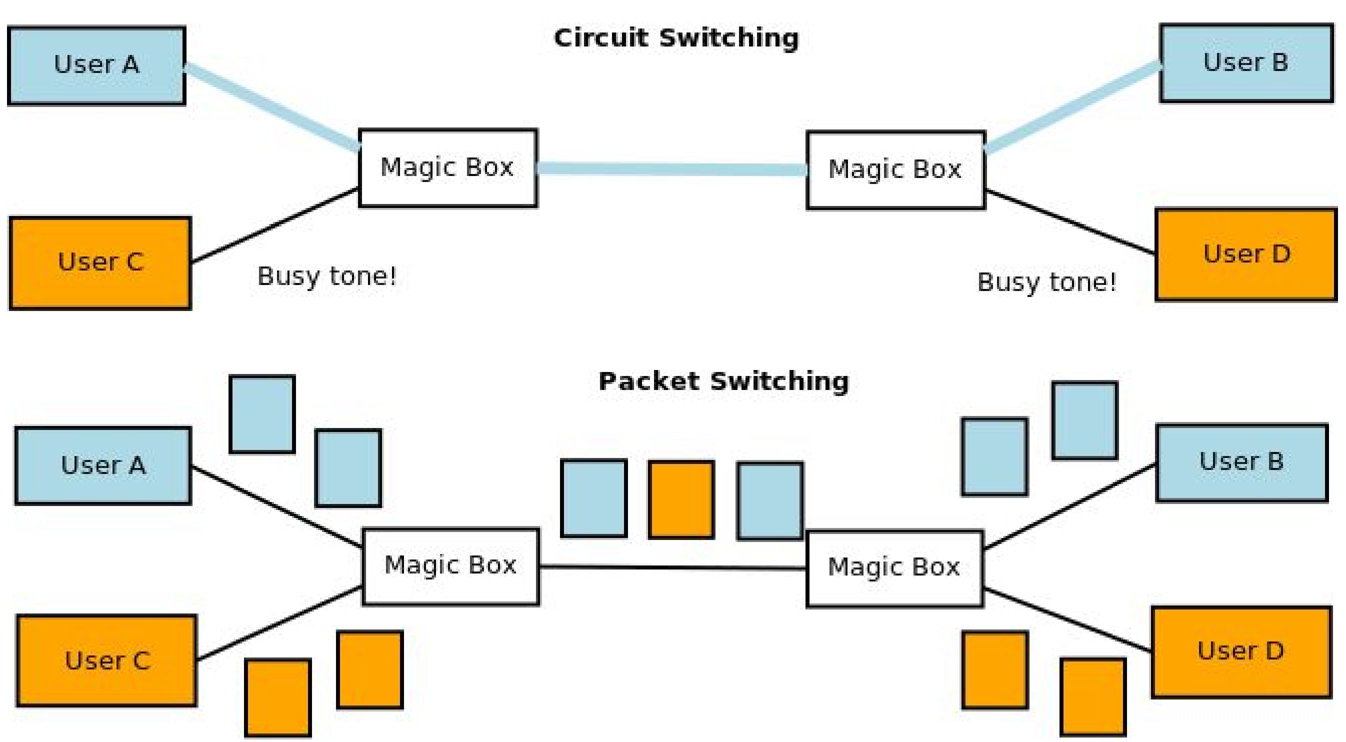
About This Talk

- Follows a real story
- The battle is not over yet
- All simulations are reproducible

Your feedback will make it better. Try it yourself and share:

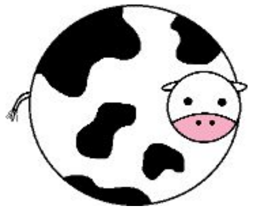
https://github.com/marchukov/talk-network-retries

Why Do We Care? Overbooking in TCP/IP Networks

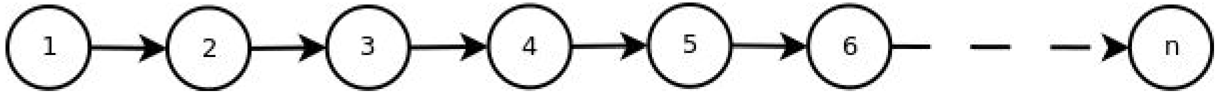


Occasional network “failures” are not failures, but “as designed” behaviour.

Why Do We Care? Rare is Not Always Rare



Chain with probability of failure of 1-part **f**:



Probability of **n**-parts chain success: $S = s^n = (1-f)^n$

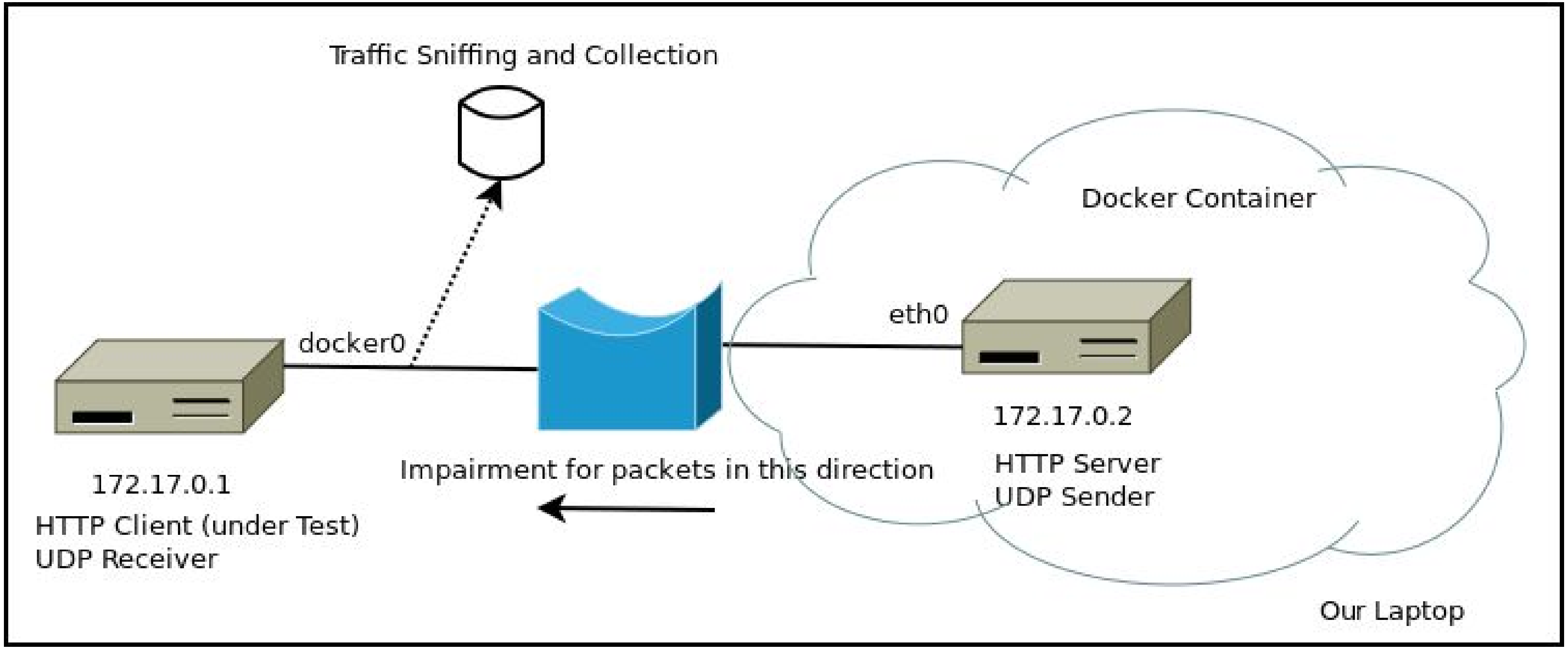
Success of **k** repetitions of **n**-parts chain: $S_k = S^k = (1-f)^{nk}$

This “amplify” rare failures:

f	n	k	S(k) ≈
0.000001	1	1	1
0.000001	100	1000	0.90
0.000001	100	10000	0.37

A spherical cow metaphor https://en.wikipedia.org/wiki/Spherical_cow. Cow image from http://abstrusegoose.com/406 under CC by-nc-sa license.

Test Environment Setup (Virtual)



Test Setup: HTTP Server with Test JSON File

```
mkdir -p ~/tmp/webroot

vi ~/tmp/webroot/test.json # Put random json (around 7 KB)

docker run --name nginx-test -v ~/tmp/webroot:/usr/share/nginx/html:ro
--privileged -d nginx

docker inspect nginx-test | grep IPAddress # "IPAddress": "172.17.0.2",
```

Now the test json file is exposed over HTTP: http://172.17.0.2/test.json

Test Setup: Network UDP Probe Using netcat

```
# Probe Receiver
ip addr | grep docker0 # inet 172.17.0.1/16 scope global docker0
nc -l -u -p 65535 > /dev/null
```

```
# Probe Sender
docker exec -i -t nginx-test apt-get -y install netcat
docker exec -i -t nginx-test bash -c 'cat /dev/urandom | nc -u
172.17.0.1 65535'
```

Capturing with WireShark (dumpcap / tshark)

```
sudo dumpcap -i docker0 -w /tmp/traffic.pcap -s 100 -f 'host
172.17.0.2'

tshark -r /tmp/traffic.pcap -T fields -E separator=, -e _ws.col.Time -e
_ws.col.Length udp.port eq 65535 > naive_probe.csv

tshark -r /tmp/traffic.pcap -T fields -E separator=, -e _ws.col.Time -e
_ws.col.Length tcp.port eq 80   > naive_download.csv

# Now we have CSV files we can load into Octave and play with
head -n 1 naive_download.csv
0.000000000,74
# _ws.col.Time, _ws.col.Length
```

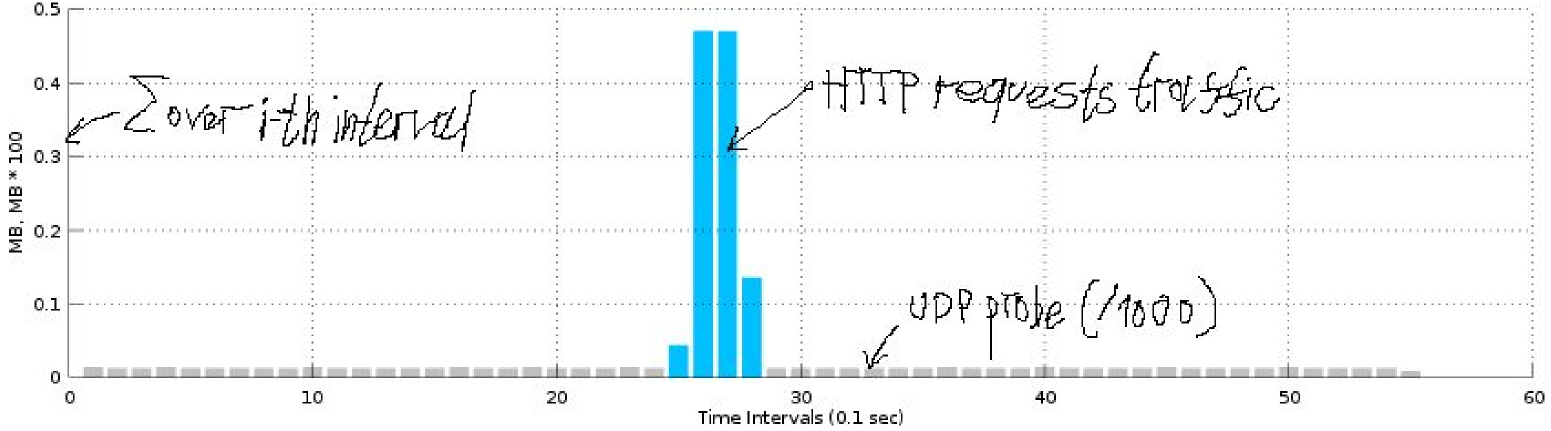
GET JSON: Naïve

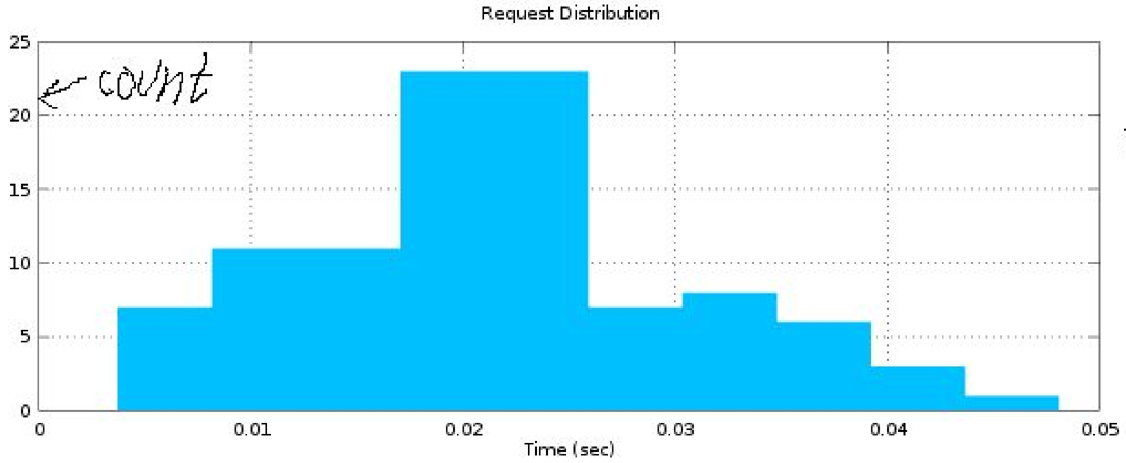
```
import requests

URL = 'http://172.17.0.2/test.json'

r = requests.get(URL)
r.raise_for_status()

res = r.json()
```

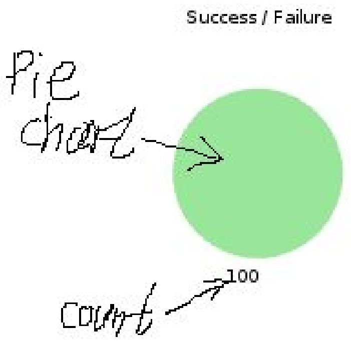
Sampler: Repeat Module Method N Times

```
# Run 100 times in a thread pool of 10 and output CSV statistics
./sampler.py 100 10 get_json naive_get > naive_get.csv



head -n 1 naive_get_json.csv
0,0.009381771087646484
0,0.0030426979064941406
0,0.002211332321166992
# Success flag (0 - ok, 1 - error), run time in seconds
```

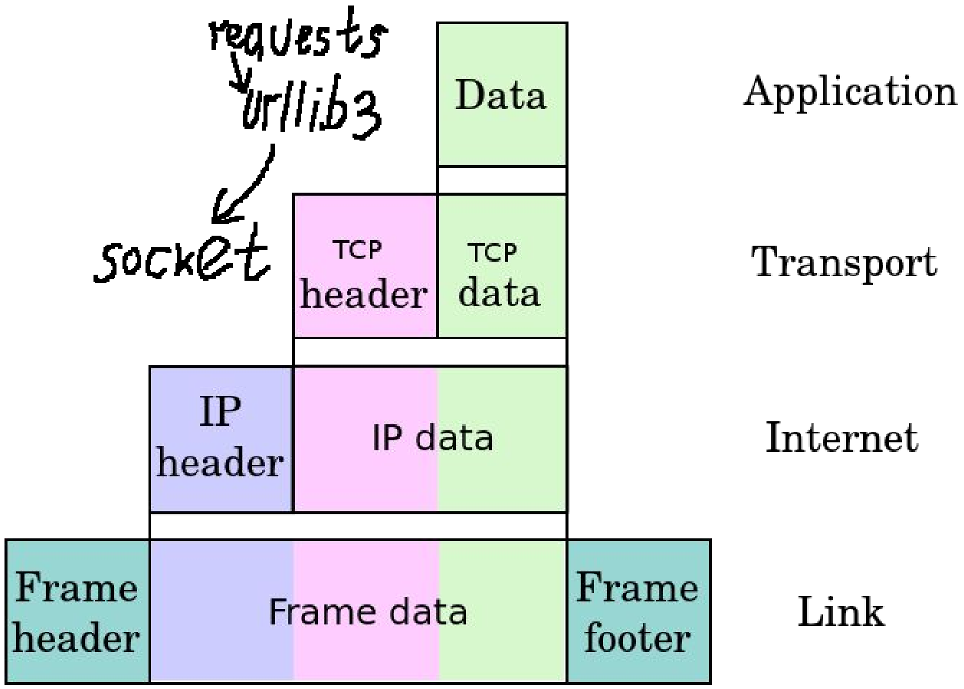
100 x 7 kB GET and Ideal Network







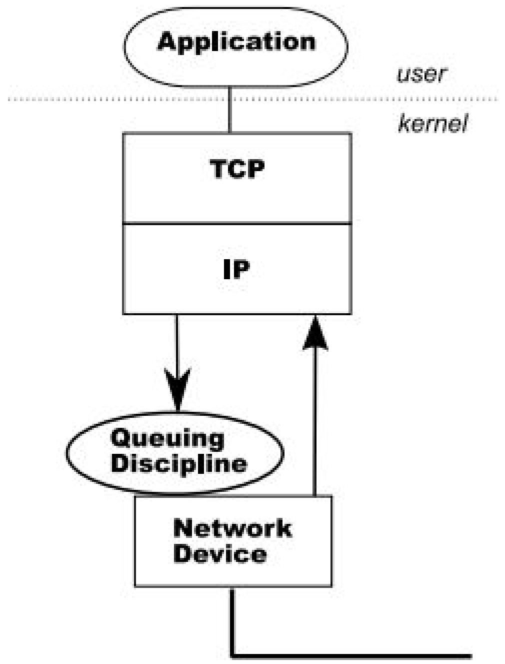
Simulation Scope and Strategy



1. Test HTTP GET request code with NetEm simulated network.
2. All failures below Python will look to us as:
   a. Data coming
   b. No data coming
   c. We get an exception
3. No library hacking.

Encapsulation diagram https://en.wikipedia.org/wiki/Internet_protocol_suite#/media/File:UDP_encapsulation.svg under CC-BY-SA license

Linux Network Emulator (NetEm)



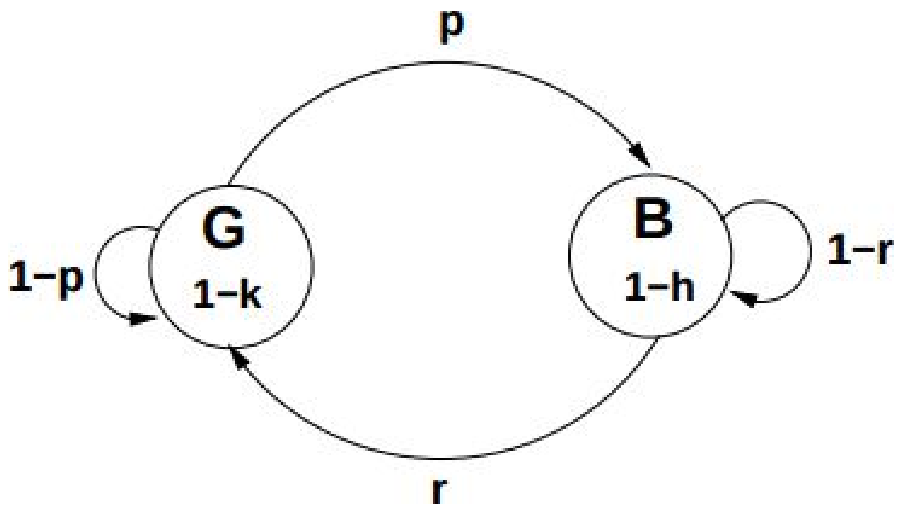
Current impairment capabilities:

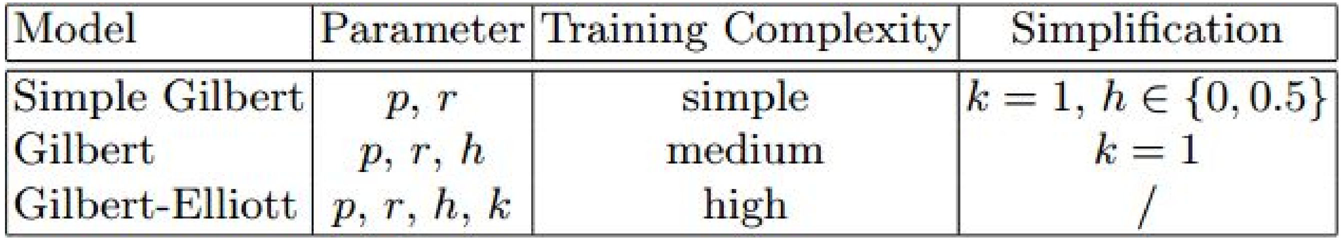
- Delay
- **Loss - we choose just this**
- Corrupt
- Duplicate
- Reorder
- Rate

Applied to **outgoing packets only**.

Image from Hemminger S. "Network Emulation with NetEm", Open Source Development Lab, April 2005.
NetEm https://wiki.linuxfoundation.org/networking/netem
Man netem(8) (usually part of iproute2 package)

Gilbert-Elliott Loss Model


$$\begin{tikzcd} \boxed{G \; 1-k} \arrow[loop left, "1-p"] \arrow[r, bend left, "p"] & \boxed{B \; 1-h} \arrow[loop right, "1-r"] \arrow[l, bend left, "r"] \end{tikzcd}$$



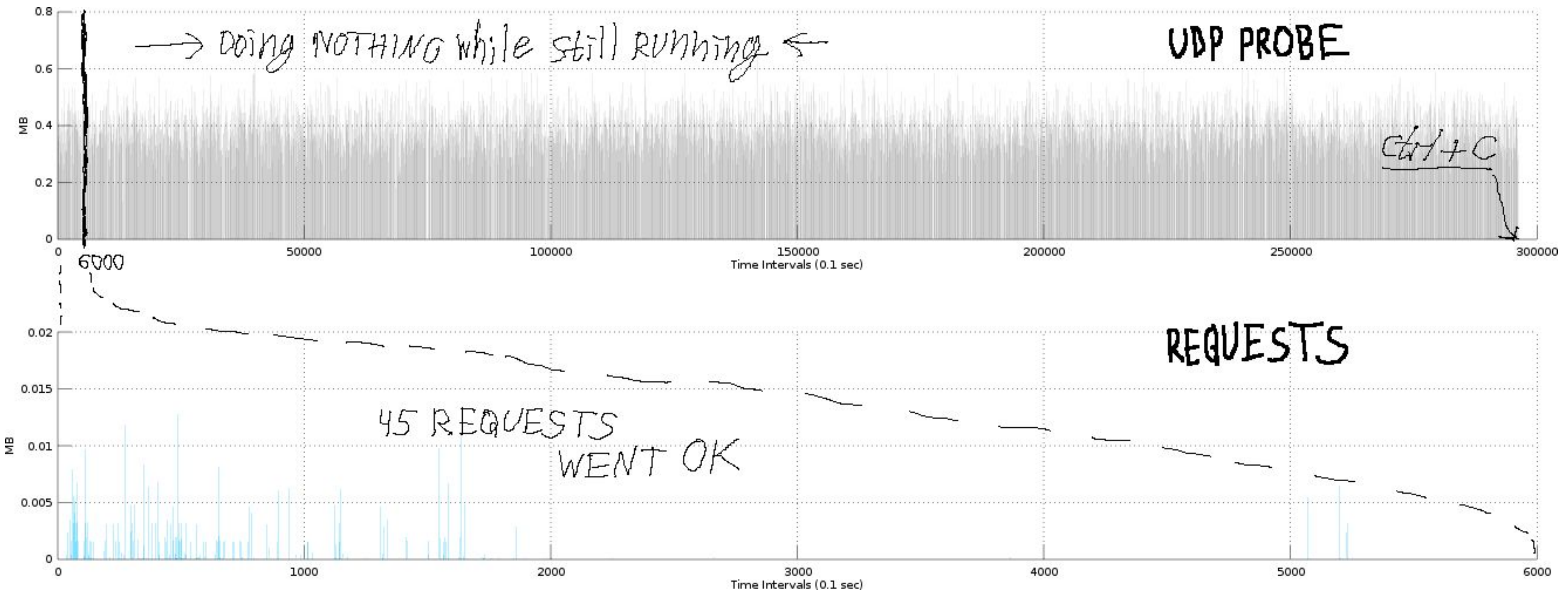
Model	Parameter	Training Complexity	Simplification
Simple Gilbert	p, r	simple	$k = 1$, $h \in \{0, 0.5\}$
Gilbert	p, r, h	medium	$k = 1$
Gilbert-Elliott	p, r, h, k	high	/

From G.Hassingler, O.Hohlfeld. The Gilbert-Elliott Model for Packet Loss in Real Time Services on the Internet. Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB), 2008 14th GI/ITG Conference

Setting Up an Impairment Using tc

```
# Inside our nginx container (that should run as privileged):

# To add
sudo tc qdisc add dev eth0 root netem loss gemodel 50 20

# To show
sudo tc qdisc show dev eth0
qdisc netem 8001: root refcnt 2 limit 1000 loss gemodel p 50% r 20% 1-h 100%
1-k 0%

# To change when it is added previously
sudo tc qdisc change dev eth0 root netem loss gemodel 50 20
```

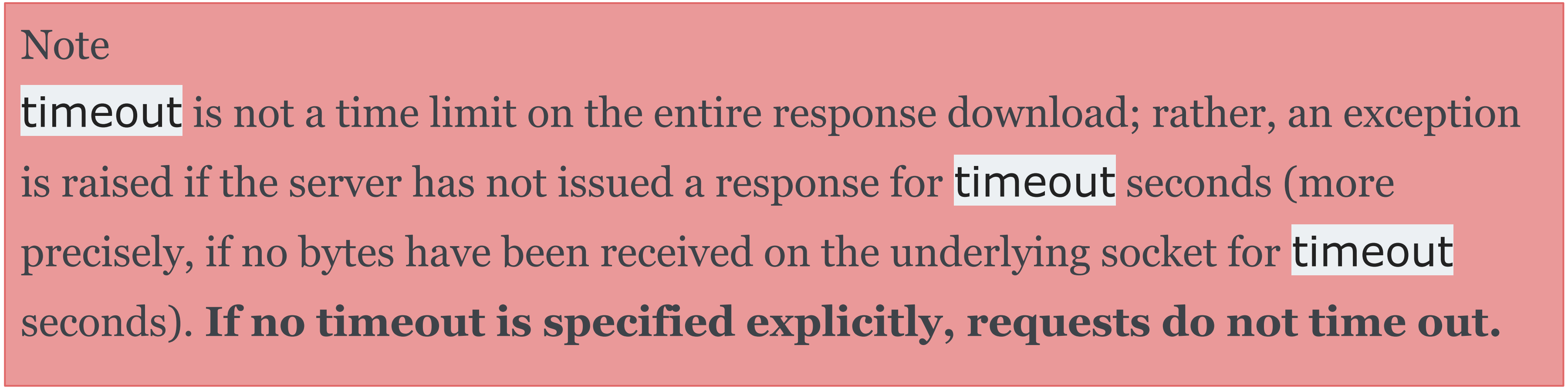
7 kB GET Run Overnight with Gilbert Loss (0.5, 0.2)



45 requests at first 600 sec then stucked. UDP was fine.

Missing Timeout: Great Way Not to Fail

And also **do nothing** over **long period** of time…



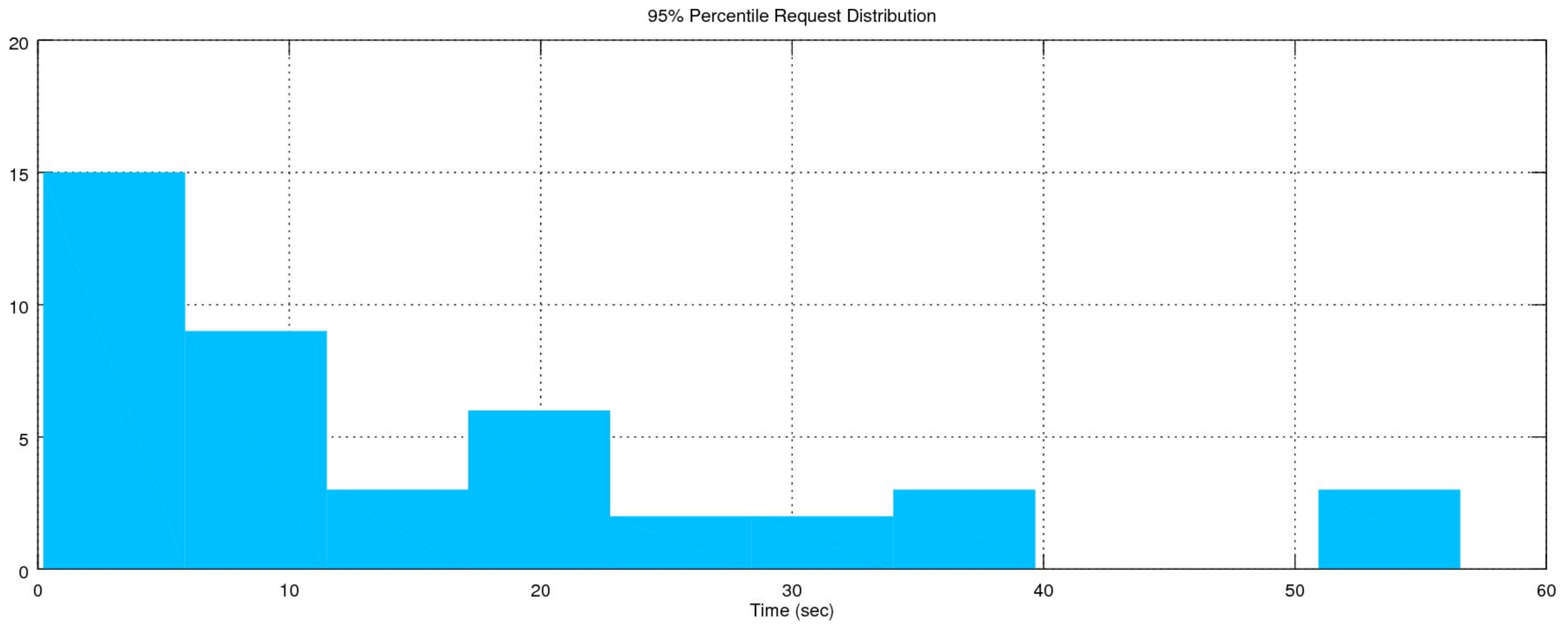
Note

`timeout` is not a time limit on the entire response download; rather, an exception is raised if the server has not issued a response for `timeout` seconds (more precisely, if no bytes have been received on the underlying socket for `timeout` seconds). **If no timeout is specified explicitly, requests do not time out.**

Do you know your required **Service Level**?

Requests 2.11.1 documentation. Quickstart. *http://docs.python-requests.org/en/latest/user/quickstart/#timeouts*

45 Out of 100 Requests Managed to Finish

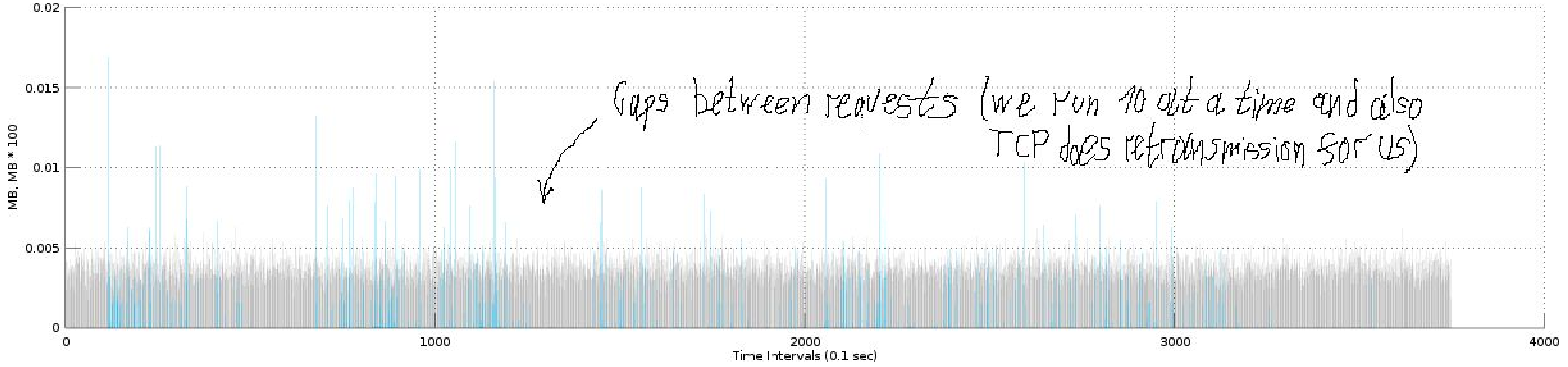


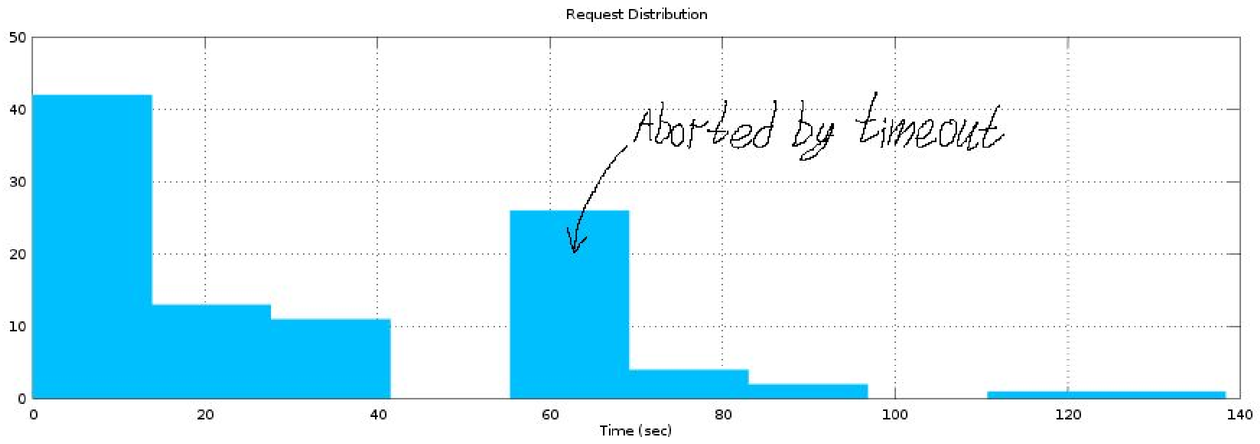
Showing requests within **95% percentile**. They all finished within **60** seconds.

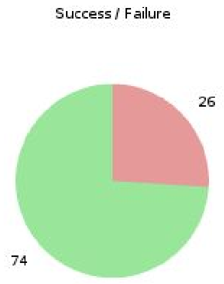
GET JSON: Less Naïve (with Timeout)

```
TIMEOUT = 60  # Seconds

r = requests.get(URL, timeout=TIMEOUT)
```

100 x 7 kB GET with G(0.5, 0.2) and Timeout 60 Sec

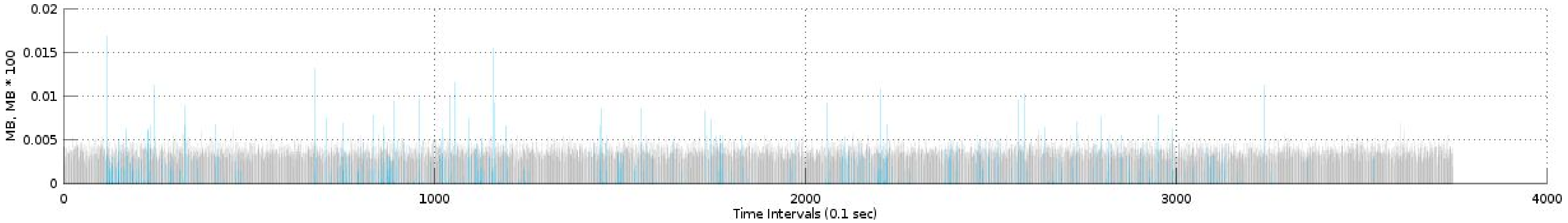






Does It Make Sense to Retry?

74 request were lucky. We try luck more and increase success probability: P(A or B) = P(A) + P(B)



TCP retransmissions do work, but will not help with:

- HTTP specific failures (not simulated).
- Failures when connection is not established (e.g. DNS errors, no route to host, etc).

Is It Safe To Retry?

General case:

- **Safe requests**
- **Idempotent requests**
- **Nothing happened**

Our case:

- **HTTP standard** defines it. Requests uses urllib3 library with retry for **any RFC compliant HTTP service**.

```
# Idempotence example

A = 1

def set_a(value):
    global A
    A = value

A          # 1
set_a(2) # 2
set_a(2) # 2
set_a(2) # 2
# …        # 2
```

Kevin Burke. A look at the new retry behavior in urllib3. https://kev.inburke.com/kevin/urllib3-retries/
What are idempotent and/or safe methods? REST Cookbook. http://restcookbook.com/HTTP%20Methods/idempotency/

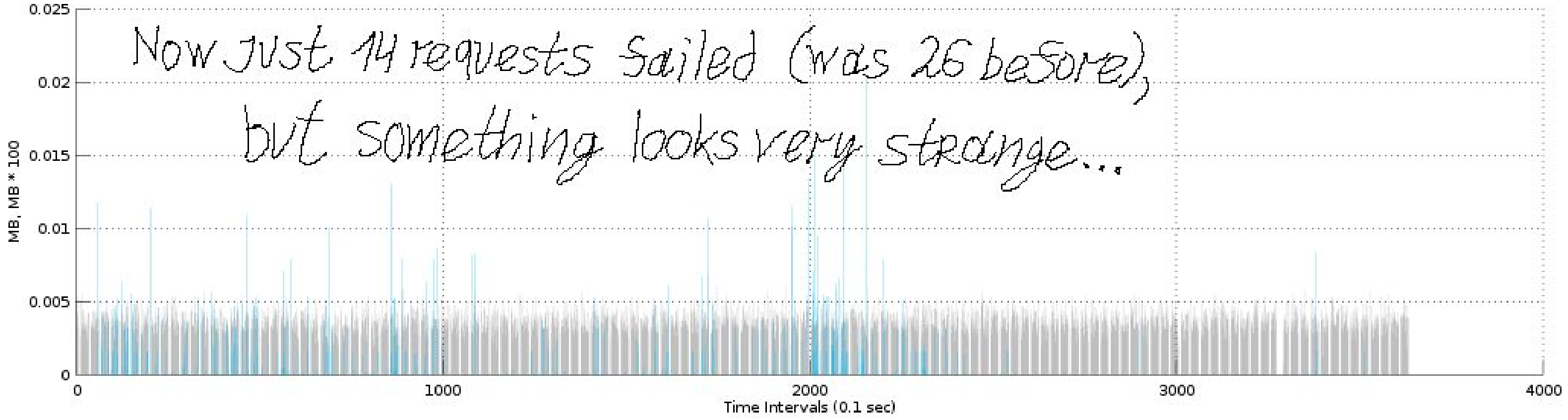
Retry Support in Python HTTP Libraries

Library	Included?	Retry?	Comments
http	Yes	No	
urllib	Yes	No	Same for urllib2
urllib3	No	Yes	New behaviour merged on Jul 2, 2014. Best I’ve found
requests	No	Yes	Uses urllib3, does not yet expose all functionality
Your Library	?	?!	Something to consider

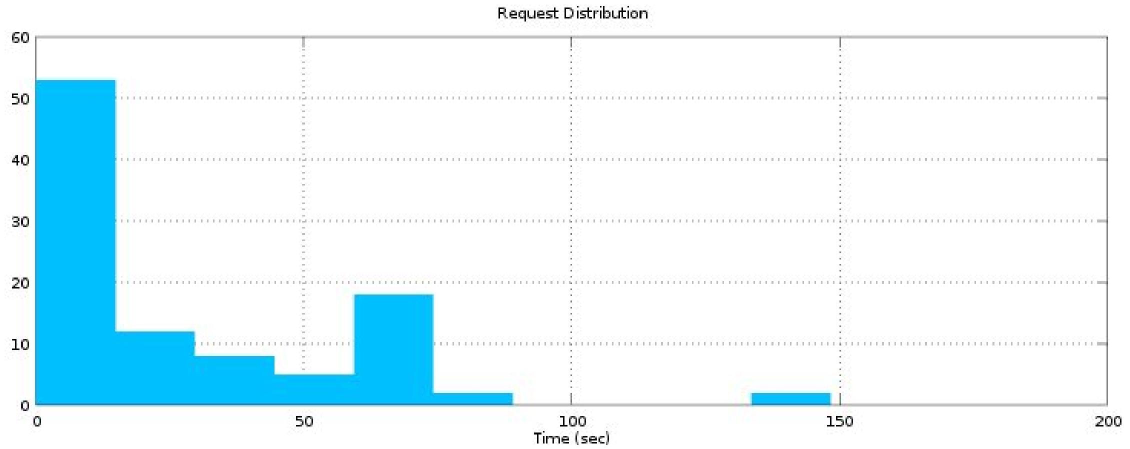
GET JSON: with Retry

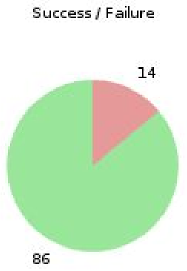
```
RETRY_PREFIX = 'http://'  # Protocol to retry
MAX_RETRIES = 3  # Number of retries

session = requests.Session()
adapter = requests.adapters.HTTPAdapter(max_retries=MAX_RETRIES)
session.mount(RETRY_PREFIX, adapter)

r = session.get(URL, timeout=TIMEOUT)
r.raise_for_status()

res = r.json()
```

100 x 7 kB GET with G(0.5, 0.2) and 3 Retries

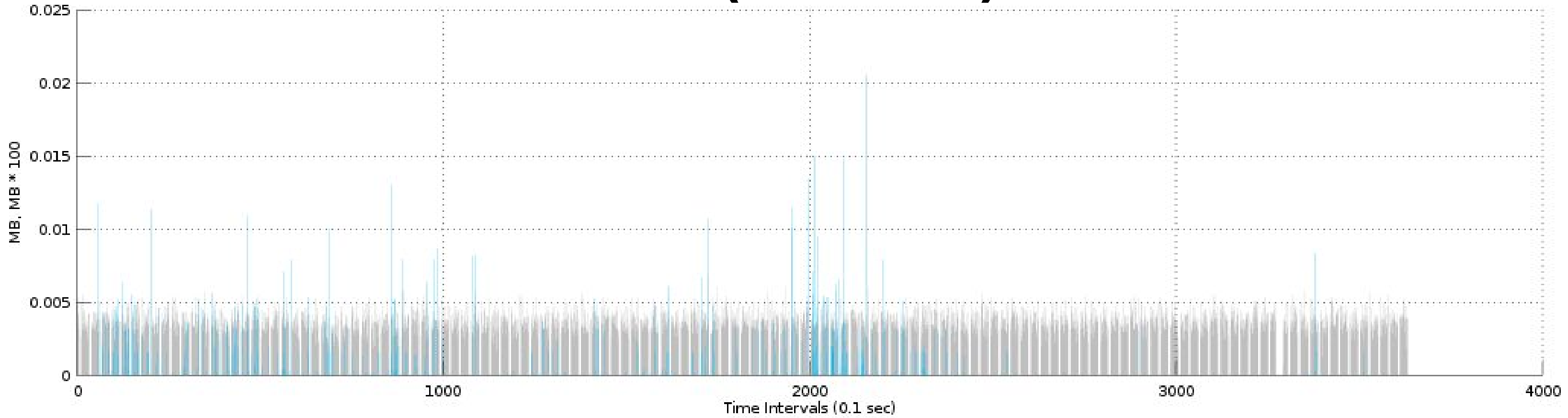


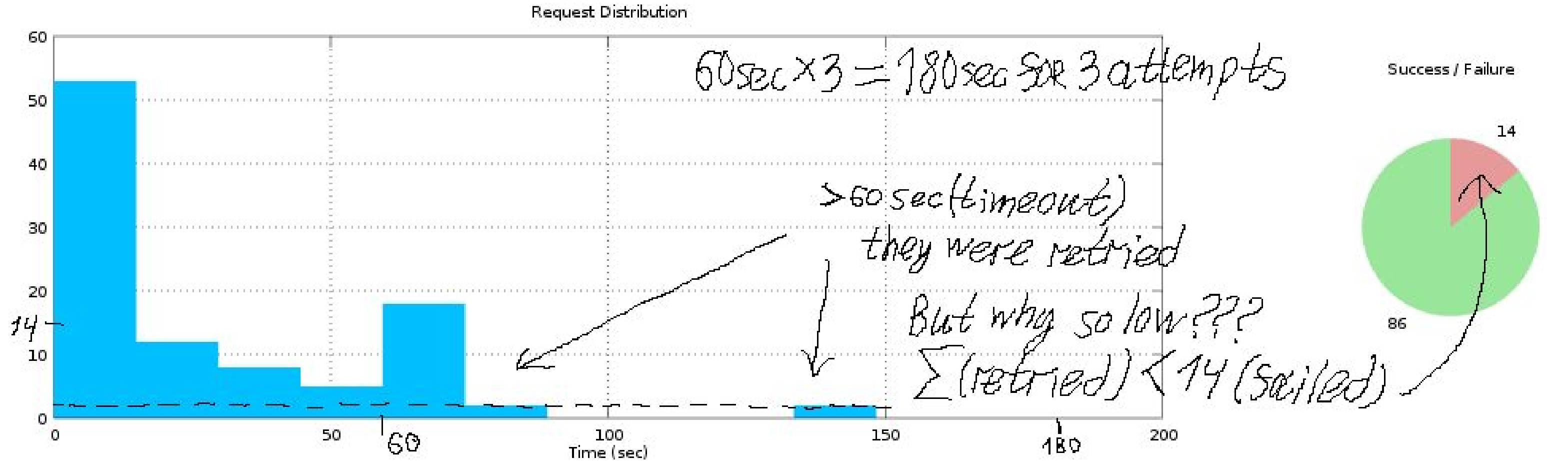
Now just 14 requests failed (was 26 before), but something looks very strange...





100 x 7 kB GET with G(0.5, 0.2) and 3 Retries





Let’s See What It Does: Enable Protocol Debug

```
import http
import logging

logging.basicConfig()
logging.getLogger().setLevel(logging.DEBUG)

http.client.HTTPConnection.debuglevel = 1

requests_logger = logging.getLogger('requests.packages.urllib3')
requests_logger.setLevel(logging.DEBUG)
requests_logger.propagate = True
```

Switch Off The Network in Test Environment

Gilbert-Elliott model with loss probability in bad **B** state **1 - k = 1**

This makes **100%** loss in both states - no network at all.

```
sudo tc qdisc change dev eth0 root netem loss gemodel 50 20 100 100
sudo tc qdisc show dev eth0
qdisc netem 803c: root refcnt 2 limit 1000 loss gemodel p 50% r 20% 1-h 100%
1-k 100%
```

Now let’s see how retry works.

Does Not Look Like It Works At All...

```
./get_json.py

DEBUG:requests.packages.urllib3.util.retry:Converted retries value: 3
-> Retry(total=3, connect=None, read=None, redirect=None)
INFO:requests.packages.urllib3.connectionpool:Starting new HTTP
connection (1): 172.17.0.2
# ...
OSError: [Errno 113] No route to host

# ... and more tracebacks below … but no traces of any new
# connection attempts
```

urllib3 Retry Object (Encapsulates HTTP Retry Behaviour)

```
retries = Retry(connect=5, read=2, redirect=5)
http = PoolManager(retries=retries)
response = http.request('GET', 'http://example.com/')
```

total	Total number. Takes precedence.
connect	Errors raised before the request is sent.
read	Errors are raised after the request was sent.
redirect	How many redirects to perform.

Kevin Burke. A look at the new retry behavior in urllib3. https://kev.inburke.com/kevin/urllib3-retries/
urllib3.util.retry documentation. http://urllib3.readthedocs.io/en/latest/reference/urllib3.util.html#module-urllib3.util.retry

Is It Safe To Retry Using urllib3 Retry Object?

1. **Disabled by default**.
2. **connect**: did not reach remote server.
3. **read**: may have side-effects.
4. **method_whitelist**: idempotent: DEFAULT_METHOD_WHITELIST = frozenset(['**HEAD**', '**GET**', '**PUT**', '**DELETE**', '**OPTIONS**', '**TRACE**']).
5. **status_forcelist**: force a retry on status: **Payload Too Large**, **Too Many Requests**, **Service Unavailable**.

urllib3.util.retry documentation. http://urllib3.readthedocs.io/en/latest/reference/urllib3.util.html#module-urllib3.util.retry
urllib3.util.retry source code. https://github.com/shazow/urllib3/blob/master/urllib3/util/retry.py

GET JSON: With Fixed Retry

```
MAX_RETRIES = 3  # Number of retries

session = requests.Session()
retry = urllib3.util.Retry(total=MAX_RETRIES,
                           connect=MAX_RETRIES,
                           read=MAX_RETRIES)
adapter = requests.adapters.HTTPAdapter(max_retries=retry)
session.mount(RETRY_PREFIX, adapter)

r = session.get(URL, timeout=TIMEOUT)
```

Still Does Not Work! Although Now It Does Retry

```
./get_json.py
INFO:requests.packages.urllib3.connectionpool:Starting new HTTP
connection (1): 172.17.0.2
# ... skipped ...
Failed to establish a new connection: [Errno 113] No route to host',)':
/test.json
INFO:requests.packages.urllib3.connectionpool:Starting new HTTP
connection (2): 172.17.0.2
# ... skipped two more connection attempts ...
# ... skipped traceback …
OSError: [Errno 113] No route to host
# ... no more retries below. It just fails … and fails all attempts quite fast
in fact ...
```

Just Kidding. We Switched the Network Off

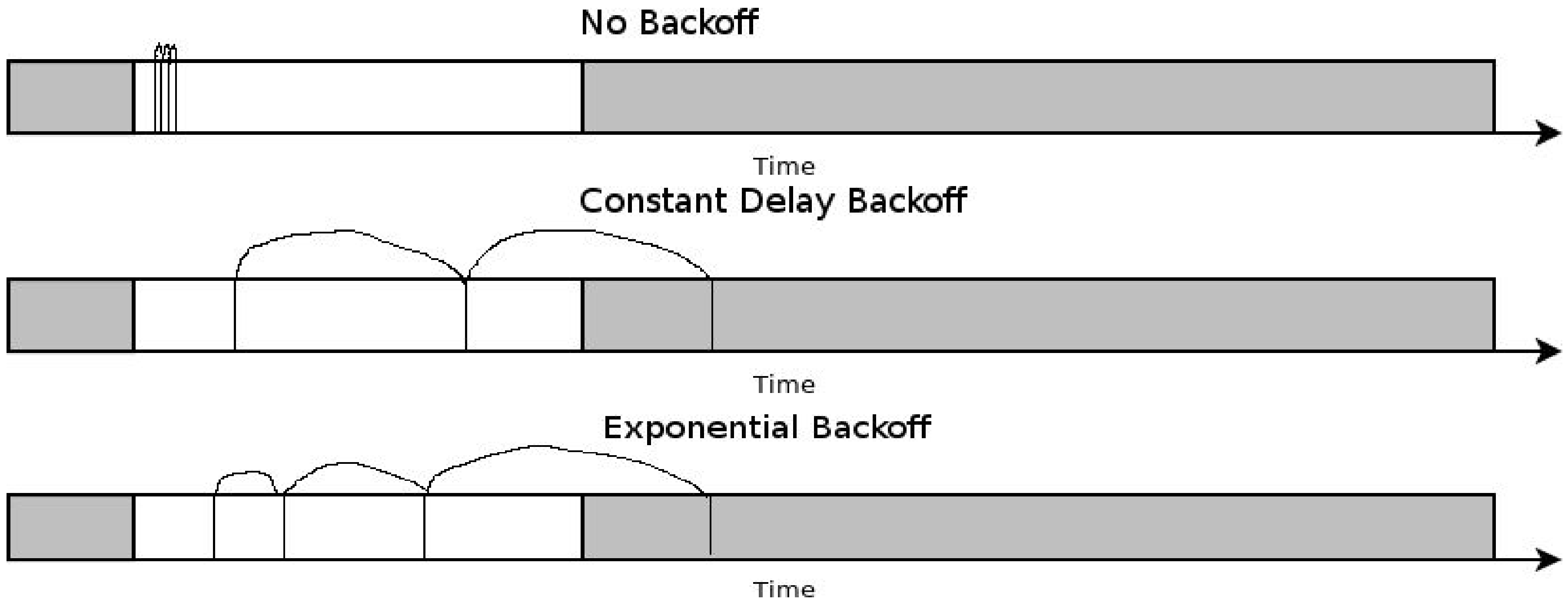
```
qdisc netem 803c: root refcnt 2 limit 1000 loss gemodel p 50% r 20% 1-h 100% 1-k 100%
```

But… Wait a minute…

Can it happen in real life too?

Yes.

Missing Backoff: Great Way to Retry and Do Not Retry



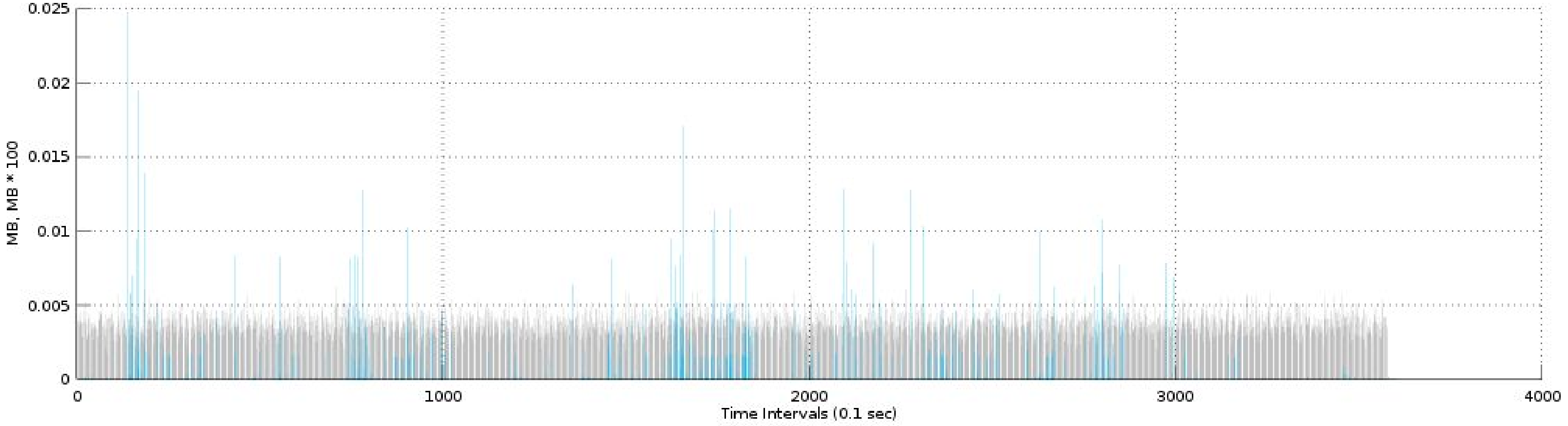
```
# For backoff_factor=1: 0 1 2 4 8 ...
backoff_value = self.backoff_factor * (2 ** (consecutive_errors_len - 1))
```

urllib3.util.retry source code. *https://github.com/shazow/urllib3/blob/master/urllib3/util/retry.py*
Exponential Backoff And Jitter. AWS Architecture Blog. *https://www.awsarchitectureblog.com/2015/03/backoff.html*

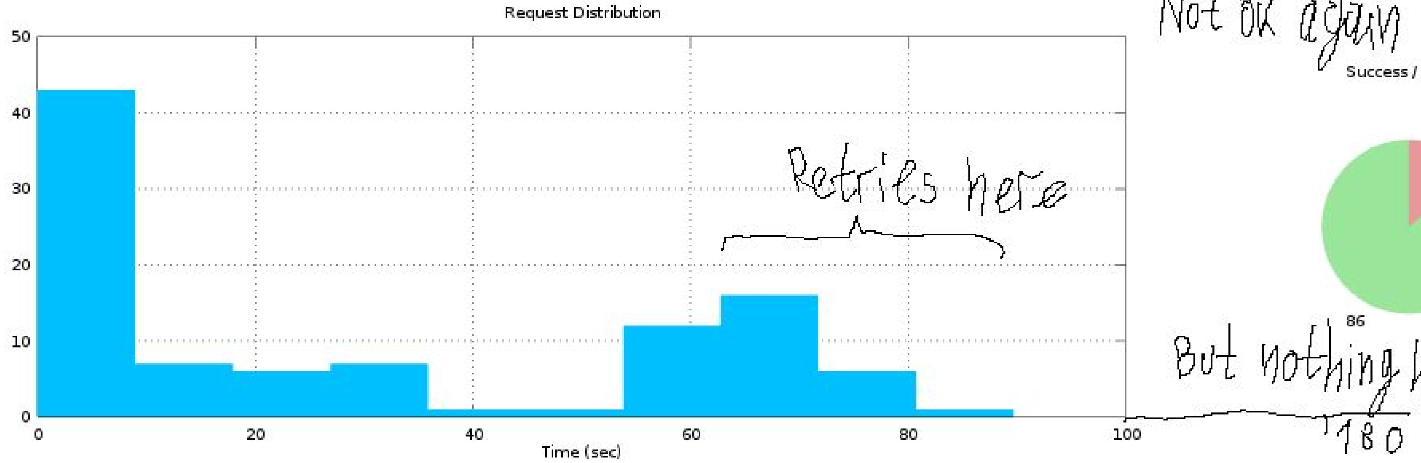
GET JSON: With Backoff Factor 25 sec (25% of Timeout)

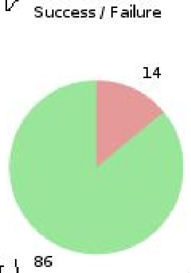
```
BACKOFF_FACTOR = 25  # Seconds

retry = urllib3.util.Retry(total=MAX_RETRIES,
                           connect=MAX_RETRIES,
                           read=MAX_RETRIES,
                           backoff_factor=BACKOFF_FACTOR)

adapter = requests.adapters.HTTPAdapter(max_retries=retry)
```

7 kB GET with G(0.5, 0.2), 3 Retries and Backoff



Not ok again :-(





Read Timeout Exceptions: Handled and Unhandled

```
WARNING:requests.packages.urllib3.connectionpool:Retrying (Retry(total=2, connect=3, read=2, redirect=None)) after connection broken by 'ReadTimeoutError("HTTPConnectionPool(host='172.17.0.2', port=80): Read timed out. (read timeout=60)",)': /test.json
```

```
ERROR:root:ConnectionError(ReadTimeoutError("HTTPConnectionPool(host='172.17.0.2', port=80): Read timed out.",),)
# ... skipped ...
  File "/usr/lib/python3/dist-packages/requests/models.py", line 737, in content
    self._content = bytes().join(self.iter_content(CONTENT_CHUNK_SIZE)) or bytes()
  File "/usr/lib/python3/dist-packages/requests/models.py", line 667, in generate
    raise ConnectionError(e)
```

GET JSON: With Our Own Retry

```
MAX_RETRY = urllib3.util.Retry(...)
def attempt(url, retry=MAX_RETRY): # Retry() is immutable
    try:
        #  … skipped session creation and passing retry to HTTPAdapter
        #   this will create new connection pool per each call :-(
        r = session.send(req, timeout=TIMEOUT)
#       except MaxRetryError:
#           raise
    except ConnectionError as e:
        retry = retry.increment(req.method, url, error=e) # return a new Retry()
        retry.sleep() #  backoff is happening here
        return attempt(url, retry=retry)
    return r
res = attempt(URL).json()
```

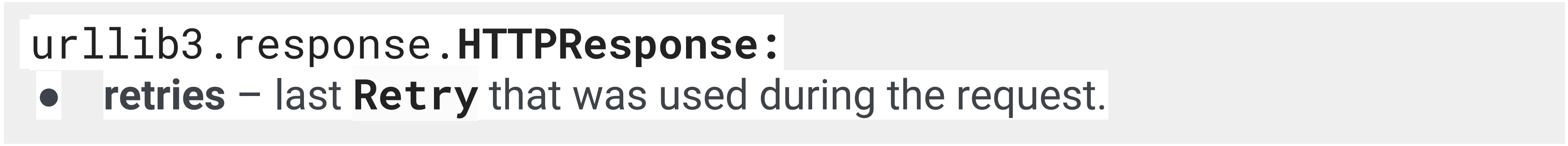
urllib3 Retry Object in Response

Previous code can retry at maximum:

MAX_RETRIES * MAX_RETRIES > MAX_RETRIES

Latest urllib3 (**not yet requests**) passes Retry() used as part of the response:

```
try:
    # ... skipped ...
    adapter = requests.adapters.HTTPAdapter(max_retries=retry)
    # ... skipped ...
except ConnectionError as e:
    retry = r.raw.retries if r else retry
    retry = retry.increment(req.method, url, error=e)
```



`urllib3.response.HTTPResponse:`

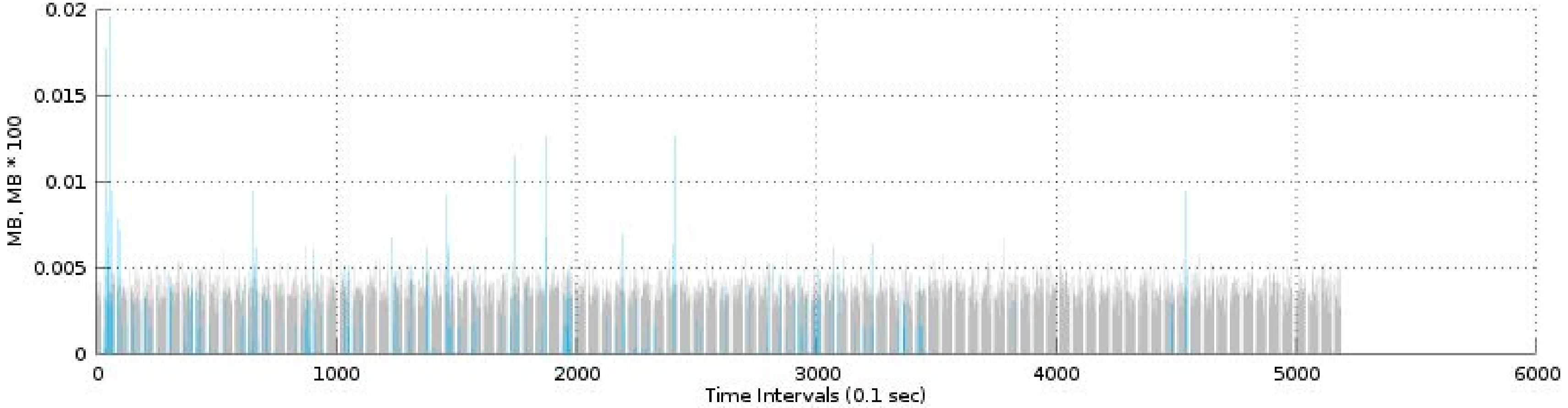
- **retries** – last `Retry` that was used during the request.

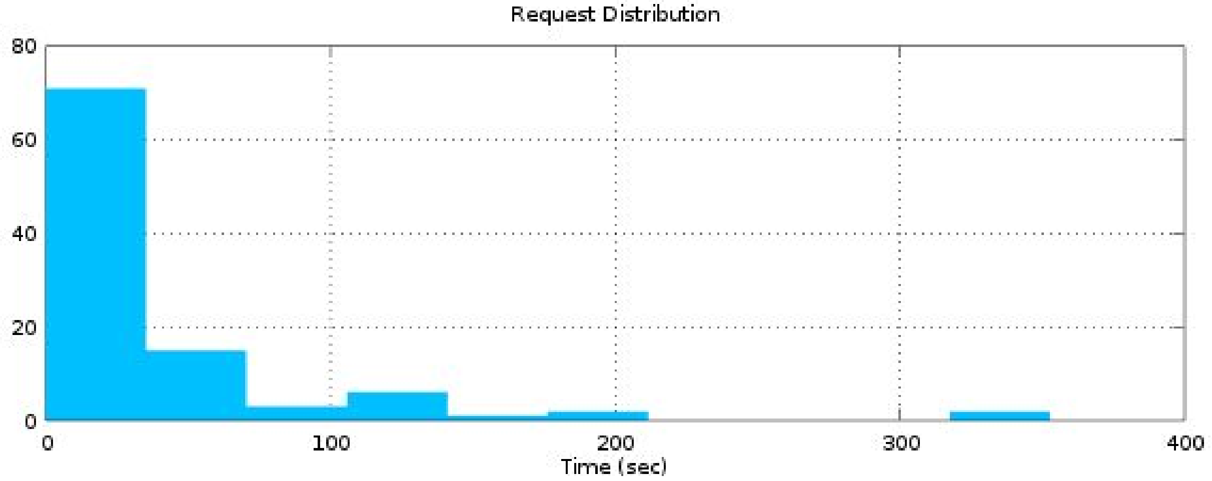
urllib3.response.HTTPResponse documentation. https://urllib3.readthedocs.io/en/latest/reference/index.html#module-urllib3.response

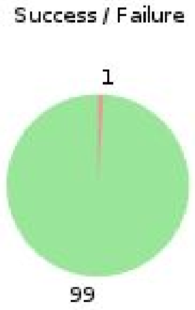
urllib3 Even Allows to Set Retry Per Request

```
import urllib3


retry = urllib3.util.Retry(...)
http = urllib3.PoolManager(retries=retry, timeout=TIMEOUT)


try:
    r = http.request('GET', url, retries=retry)
except ...
```

7 kB GET with G(0.5, 0.2) and Our Own Retry







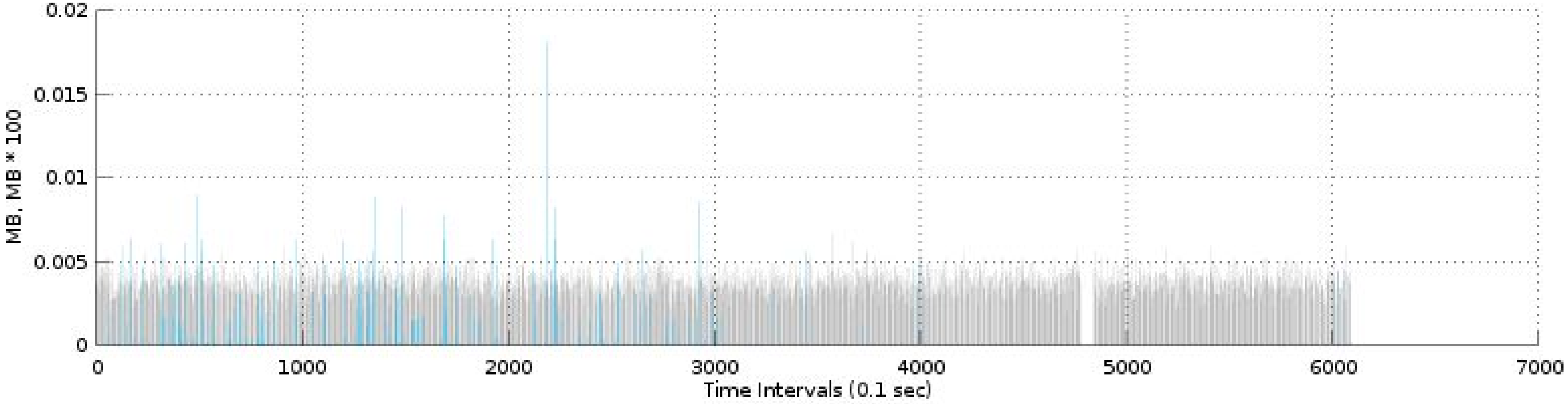
Still 1 Request Failed. Can We Do Even Better?

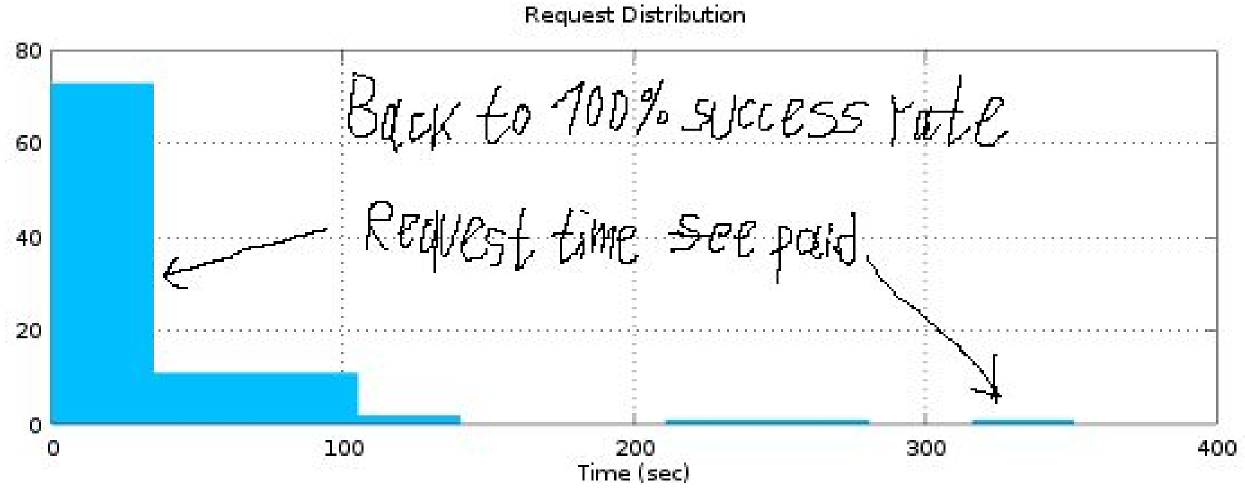
```
ERROR:root:object of type 'NoneType' has no len()

File "/usr/lib/python3/dist-packages/requests/models.py",
line 791, in json
    if not self.encoding and len(self.content) > 3:
```

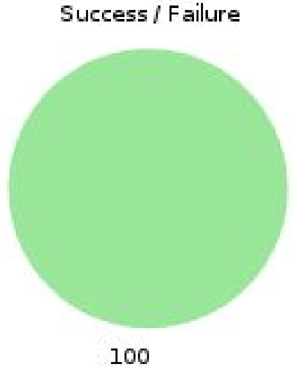
GET JSON: Retry With Content Awareness

```
def attempt(url, retry=retry):
    try:
        # … skipped …
        r = session.send(req, timeout=TIMEOUT)
        r.raise_for_status()
        j = r.json()
    # DEMO ONLY. TypeError is too wide to handle here
    except (ConnectionError, TypeError) as e:
        retry = retry.increment(req.method, url, error=e)
        retry.sleep()
        return attempt(url, retry=retry)
    return j
res = attempt(URL)
```

7 kB GET with G(0.5, 0.2) and Content Aware Retry







Conclusion

1. We can emulate network good enough.
2. Testing on “localhost” network does not work.
3. Testing on local network also might not work.
4. Implementing a retry is not easy. Use existing solutions when possible.
5. If you do your network library or protocol consider standard retries built in.
6. But, provide users ability to customize and override based on their use case.

And all this is possible!

Questions?

@martchukov

anton@marchukov.com

https://github.com/marchukov/talk-network-retries