Flamethrower

Jan Včelák

February 3, 2019 at FOSDEM

NS1. ns1.com · @fcelda @ns1 · jvcelak@ns1.com

Flamethrower: DNS Performance Testing Tool

https://github.com/DNS-OARC/flamethrower

Apache Public License 2.0

Built as an alternative to *dnsperf*.

Initial requirements:

- Better source port distribution
- Solid TCP support
- Realistic query rate patterns
- Possibility to integrate into a CI/CD pipeline

NS1.

Flamethrower Quick Start

```
> flame \
  -r foo.example.test. \
  -T SOA \
  -Q 2000 \
  -l 5 \
  a.ns.example.test
```

```
flaming target "a.ns.example.test" (192.0.2.100) on
port 53 with 10 concurrent generators, each sending
10 queries every 1ms on protocol udp
query generator [static] contains 1 record(s)
----------------------------------------------------
1.87965s: send: 2001, avg send: 2877, recv: 1581,
avg recv: 2240, min/avg/max resp:
23.4206/17.1624/52.33ms, in flight: 1285, timeouts:
0
----------------------------------------------------
runtime      : 7.88192 s
total sent   : 11756
total rcvd   : 9223
min resp     : 23.3888 ms
avg resp     : 27.9633 ms
max resp     : 33.3626 ms
avg rps      : 1834
avg qps      : 2350
avg pkt      : 31.92 bytes
timeouts     : 2533 (21.5464%)
responses    :
  SERVFAIL: 441
  NOERROR: 8782
```

NS1.

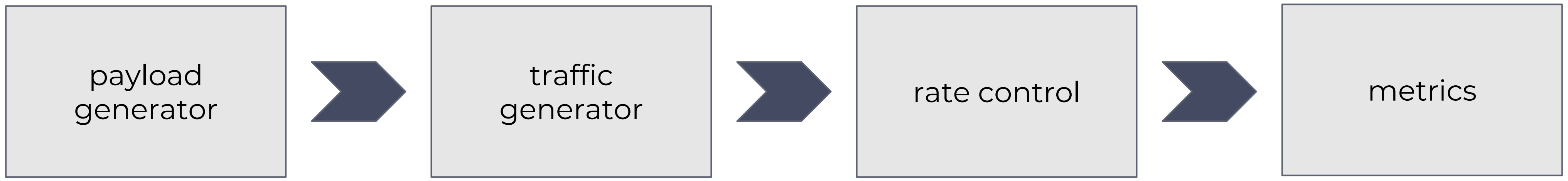
Flamethrower Internals

Flamethrower is written in C++14 (17).

Notable dependencies:

- LDNS - DNS library
  https://nlnetlabs.nl/projects/ldns/about/
- uvw - asynchronous I/O (modern C++ wrapper for libuv)
  https://github.com/skypjack/uvw

Flamethrower Architecture



NS1.

Payload Generators

- **static** — Single query name and type.
- **file** — Consumes dnsperf-compatible input file (i.e. pairs of query name and type).
- **randompkt** — Random binary garbage.
- **numberqname** — Names prefixed with a random numeric label:
  - 1234.example.test.
  - 42.example.test.
  - ...
- **randomqname** — Names prefixed with random binary labels:
  - j4kJx\000zz3d\064sa.example.test.
  - bo4._kf\042.example.test.
  - ...
- **randomlabel** — Names prefixed with random non-binary labels:
  - XRY4HFY9dpItTb.example.test.
  - ady1.clo.ZUqV4gji.example.test.
  - ...

NS1.

Traffic Generator

Flamethrower runs in a **single thread** only.

Max query rate ~100k queries/s on a single core.

Traffic generator config options:

- **-c** Number of concurrent traffic generators.
- **-q** Number of queries to send in a batch.
- **-d** Time (delay) between batches.

Protocol selection option:

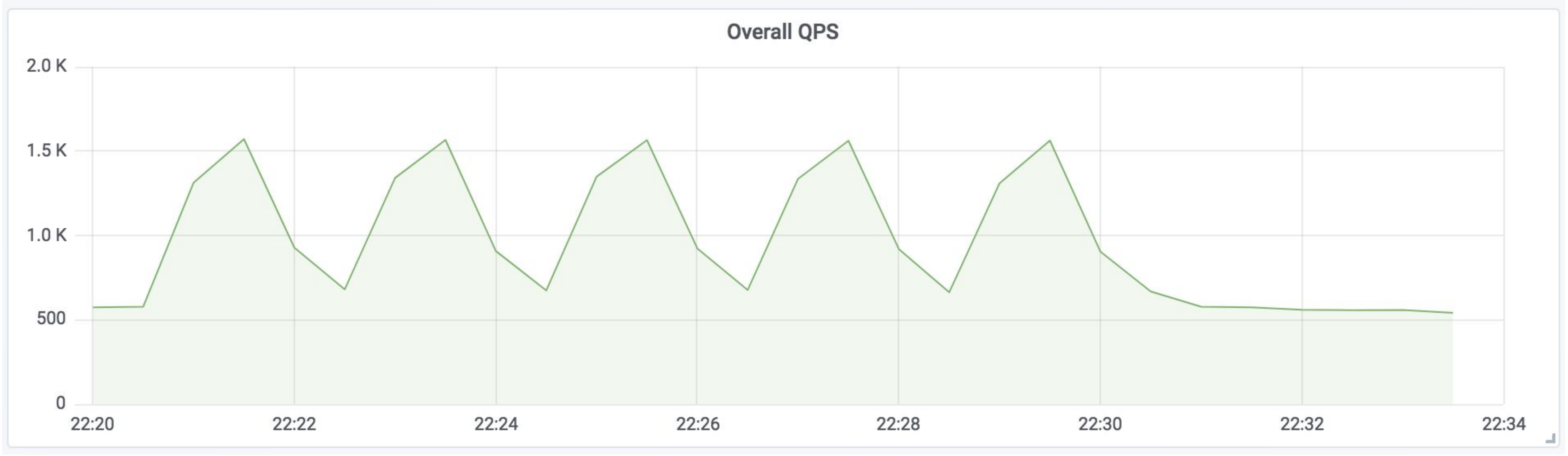
- **-P udp** Default. One bound UDP socket per traffic generator.
- **-P tcp** One TCP sessions per traffic generator. The session is re-established when the server closes the connection.

NS1.

Rate Control

Constant rate: `flame -Q 1000`

Variable rate: `flame --qps-flow '1000,60000;100,60000;...'`



Metrics

```
27.8948s: send: 1000, avg send: 1025, recv:
995, avg recv: 1018, min/avg/max resp:
63.1087/27.7805/98.1353ms, in flight: 81,
timeouts: 17

run id       : c50c74a39823a4c1
run start    : 2019-02-01T22:38:01Z
runtime      : 122.894 s
total sent   : 120887
total rcvd   : 120581
min resp     : 62.7252 ms
avg resp     : 41.2032 ms
max resp     : 94.3204 ms
avg rps      : 987
avg qps      : 998
avg pkt      : 23.4042 bytes
tcp conn.    : 0
timeouts     : 306 (0.253129%)
bad recv     : 0
net errors   : 0
responses    :
  NOERROR: 120581
```

Machine friendly variant available with:

```
> flame -o results.json
```

{"period_bad_count":0,"period_in_flight":81,"period_net_errors":0,"period_pkt_size_avg":16.4,"period_response_avg_ms":27.63856731149192,"period_response_max_ms":79.465695,"period_response_min_ms":63.179727,"period_timeouts":0,"run_id":"c50c74a39823a4c1","runtime_s":18.893585102,"total_bad_count":0,"total_net_errors":0,"total_pkt_size_avg":21.57894736842105,"total_qps_r_avg":1032,"total_qps_s_avg":1043,"total_r_count":19707,"total_response_avg_ms":42.745088149163905,"total_response_max_ms":79.465695,"total_response_min_ms":62.896545999999994,"total_responses":{"NOERROR":19707},"total_s_count":19892,"total_tcp_connections":0,"total_timeouts":114}

Samples

10 queries every 500 ms on 100 source ports over UDP (= 2000 q/s). Queries read from a dnsperf-compatible input file.

```
> flame -q 10 -d 500 -c 100 \
  -g file -f queries.txt \
  ns.example.test
```

10 queries every 250 ms on 250 source ports over UDP (= 10k q/s) with a query rate decay. Queries for a random subdomains on example.test. Metrics written into a JSON file.

```
> flame -q 10 -d 250 -c 250 \
  --qps-flow '10000,5000;5000,5000;2500,5000;1250,5000;1000,40000' \
  -g randomlabel lblsize=10 lblcount=1 count=140000 -r example.test -T AAAA \
  -o peak.json \
  ns.example.test
```

1 "query" every 100 ms on 100 concurrent IPv6 TCP sessions (= 1000 q/s). Random "garbage" up to 100 bytes.

```
> flame -q 1 -d 100 -c 100 \
  -g randompkt count=10000 size=100 \
  -F inet6 -P tcp ns.example.test
```

NS1.

Future Ideas

- Multi-processor support
- Target multiple servers (IP range)
- Source address spoofing
- DNS-over-TLS and DNS-over-HTTP support
- Query rate jitter
- Query rate pattern extracted from a pcap
- Improve performance

Hank Scorpio, The Simpsons ep. 155

Jan Včelák

@fcelda
jvcelak@ns1.com

Thank you!

https://github.com/DNS-OARC/flamethrower

NS1.

Bonus slides

NS1.

UDP echo server in libuv (part 1)

```
uv_loop_t loop = {0};
uv_loop_init(&loop);

uv_udp_t server = {0};
uv_udp_init(&server);

struct sockaddr_in6 addr = {0};
uv_ip6_addr("2001:db8::c01d:cafe", 1234, &addr);

uv_udp_bind(&server, (struct sockaddr *)&addr, UV_UDP_REUSEADDR);

uv_udp_recv_start(&server, buf_alloc, on_recv);

uv_run(&loop, UV_RUN_DEFAULT);
uv_loop_close(&loop);
```

UDP echo server in libuv (part 2)

```
struct send {
    uv_udp_send req;
    uv_buf_t buf;
};
static void buf_alloc(uv_handle_t *, size_t hint, uv_buf_t *buf) {
    buf->base = malloc(hint);
    buf->len = hint;
}
static void on_recv(uv_udp_t *server, ssize_t nread, const uv_buf_t *buf,
                    const struct sockaddr *addr, unsigned flags) {
    struct send *s = calloc(1, sizeof(*s));
    s->buf = *buf;
    uv_udp_send(req, server, &s->req, 1, addr, on_sent);
}
void void on_sent(uv_udp_send *req, int status) {
    struct send *s = (struct req *)(req);
    free(s->buf.base);
    free(s);
}
```

UDP echo server in uvw

```
std::shared_ptr<uvw::Loop> loop = uvw::Loop::getDefault();
std::shared_ptr<uvw::UDPHandle> server = loop->resource<uvw::UDPHandle>();
server->on<uvw::UDPDataEvent>([](uvw::UDPDataEvent &ev, uvw::UDPHandle &) {
    handle->send<uvw::IPv6>(ev.sender, std::move(ev.data), ev.length);
});
server->bind<uvw::IPv6>("2001:db8::c01d:cafe", 1234);
server->recv();
loop->run();
```

NS1.