Review of DNS and HTTP PM Tools

Survey of existing DNS tools

Tool	UDP	TCP	TLS	Pipelining	Uses query file	Replay pcaps	Comments
`dnsperf from DNS-OARC`	Y	N	N	N	Y	N	https://github.com/DNS-OARC/dnsperf
`resperf from DNS-OARC`	Y	N	N	N	Y	N	For testing resolvers https://github.com/DNS-OARC/dnsperf
`dnsperf-tcp`	Y	Y	N	Y	Y	N	https://github.com/Sinodun/dnsperf-tcp
`dnsperf-tls`	Y	Y	Y	Y	Y	N	https://github.com/Sinodun/dnsperf-tcp/tree/feature/tls_openssl. A re-factor was required to accomodate TLS usage within the threading model used here and we believe this introduces a performance overhead at very low queries per connection (below 500). This is being investigated.
Flamethrower	Y	Y	Y	Y	Y	N	https://github.com/DNS-OARC/flamethrower
`tcpscaler`	Y	Y	Y	Y	Y	N	 https://github.com/jonglezb/tcpscaler
`perftcpdns`	N	Y	N	N	Y - in hex format	N	A performance testing tool for DNS over TCP, available in the contrib directory of all recent versions of BIND9
`queryperf++`	Y	Y	N	N			Opensource framework for testing DNS servers that uses both UDP and TCP: https://github.com/jinmei/queryperfpp
`Drool`	Y	Y	N	?	Y	Y	`drool` can replay DNS traffic from packet capture (PCAP) files https://github.com/DNS-OARC/drool
`dns-benchmarking/`	Y	N	N	N	Y	?	https://gitlab.labs.nic.cz/knot/dns-benchmarking/tree/master
ISC Performance lab	Y	N	N	N	Y	?	https://github.com/isc-projects/perflab

Investigation of HTTP benchmarking tools

Survey

We've also looked a some tools to see if we can reuse anything from the HTTP measurement world to help with benchmarking of Dot or DoH.

As a minimum requirement a tool must be capable of making test appear to come from at least 1000 individual clients (aka *virtual users* or VU in web server performance speak) but this would still require some orchestration to reach (ideally) ~30,000 VU per test VM to minimise the total number of test VM needed.

- Tsung
  - Pro:
    - Generated traffic with 7500 VU on my Mac pretty much out of the box. According to `tsung status`, anyway.
    - Reviews say it can generate 20-30k VU per machine.
    - Written in Erlang.
    - Has existing plugins for non-HTTP traffic, e.g. LDAP, MySQL server, and support for UDP and TCP/TLS.
    - Can do basic scripting of client behaviour in XML config. Appears possible to script desired behaviour.
  - Con:
    - Mailing list is quiet (few messages every other month) and not much commit activity on GitHub.
    - No DNS plugin, and would need to write one. In Erlang.
  - **Conclusion**: worth further investigation, see below
- k6
  - Pro:
    - New. Shiny. Popular on GitHub.
    - Client scripts written in JavaScript, so can produce desired behaviour.
    - Written in Go, so good DNS lib available.
    - Can export results to InfluxDB ready for graphing by Grafana.
  - Con:

        - No non-HTTP modules as yet.
        - Does not necessarily seem to scale to large number of VU. I tried running a simple 10s test on `jim-dev-linux`. Test startup took 2s on 10 VU, 9s on 100 VU, 41s on 500 VU, 63s on 750 VU and 26m9s on 1000 VU. The latter was because it went well into swap on a 4Gb RAM VM.
    - **Conclusion**: Prohibitive startup times with 1000 VU

Others surveyed but not tested:

- Locust
    - Pro:
        - Python.
        - Client is also Python, so can produce desired behaviour.
        - Good at distributed clients.
        - Scales to a lot of clients.
    - Con:
        - In Python, so not fast. Max query rates 1/25 competitors on same hardware.
        - Relies on client scaling for volume. Aimed at those willing to spin up a lot of AWS instances.
- JMeter
    - Pro:
        - DNS howto: https://jmeter-plugins.org/wiki/dns_test_using_jmeter/
        - Widely used.
    - Con:
        - Reports says maxes out at 1000 VU per machine. Needs 1 Java thread per VU.

Work on Tsung

After further work on `Tsung` we concluded that peak traffic generation for single client instance of 30k clients was limited to 100kqps, as discussed below. This would mean many client VM's would be needed to achieve moderate DNS traffic levels. We suspect the fundamental reason behind this is twofold:

1. Each session is scripted, so you can mimic real traffic. But that scripting is at some level interpreted.
2. HTTP testing happens at a lower order of magnitude of connections and traffic than DNS. Hitting a website with 10,000 HTTP GETs down the same connection just doesn't really happen.

We did do some development work on it to add DNS as a proof of concept, the result can be found in the dns-client branch in this GitHub repo. It proved reasonably easy to get a DNS plugin doing synchronous queries working over UDP, TCP and TCP/TLS. Even taking successive lookups from a queries file as used by `dnsperf`. Our specific observations are:

- Dynamic variable substitution on each query slows down activity significantly, and according to this PR currently can be a serious bottleneck for horizontal scaling.
- We found that yes, you can create a lot of concurrent sessions (i.e. Virtual Users). However, the total throughput of queries that you can generate falls off markedly; a single session generating 7,000,000 UDP packets that it sends and doesn't check for answers can send 100k packets/sec, but increasing the number of sessions doesn't increase the overall output. Instead you rapidly find total output halving if you add several more sessions, and moving to a lot of sessions reduces total output to more like 7k. No matter what we changed, we've been unable to get total output much above 100kqps.
- Adding support for pipelining queries (having multiple queries in flight from a single session, necessary to mimic client behaviour) is not straightforward. The concept just doesn't exist in Tsung at present; it's basically synchronous only.

So, the problem is that we're unable to use `Tsung` to generate baseline performance numbers to compare with existing numbers. While it might (after some work) be useful for real-life type scenarios, there are concerns about how well it will scale horizontally (though that may well be the dynamic variable issue above). The original author does designed for many parallel runs of complex sessions - so, probably good for real life tests, but a problem if you're trying to establish a baseline of maximum queries a server can handle.