lieter_
PowerDNS

pieterlexis
PowerDNS

dnsdist: the high-performance, DoS and abuse-aware DNS loadbalancer

Pieter Lexis

November 3rd 2016

whoami

Pieter Lexis

- “PowerDNS Engineer”
- C++/Python/Shell progammer
- SysAdmin

Founded in 1999, Open Source since 2002

- Authoritative Server
  - 30% to 50% of all domains
  - >85% of all DNSSEC domains
  - “one-click” DNSSEC
- Recursor
  - 100+ million internet users
  - DNSSEC validation since 4.0.0
- Since 2015 part of Open-Xchange, together with Dovecot

Introduction and History

Importance of DNS

Everything on the Internet starts with a DNS lookup

DNS lookup slow ➜ everything slow

One of the least measured protocols on the internet

“It works for me”

Debugging? a combination of `tcpdump`, `awk`, `grep` and `dig`

dnsdist – History and Origins

```
dnsdist listen-ip dest-ip-1 dest-ip-2
```

`dnsdist` – History and Origins

```
dnsdist listen-ip dest-ip-1 dest-ip-2
```

- Most load balancers know about HTTP(S), IMAP etc.
- DNS can’t be handled as “a weird kind of web”
- Observation: A busy nameserver is a happy nameserver
- “concentrating load balancer”

Features

In A Nutshell

“dnsdist is a highly DNS-, DoS- and abuse-aware loadbalancer. Its goal in life is to route traffic to the best server, delivering top performance to legitimate users while shunting or blocking abusive traffic.”

In A Nutshell

Swiss army knife of DNS problem solving

- Add and remove flags
- Filter traffic (from the kernel)
- Inspect live traffic from the console
- Delay and rate limit bad queries
- Huge amount of statistics
- Open Source (GPLv2) and vendor neutral
- Smart load balancing

Scenarios

- Statistics for legacy nameserver platform
- Realtime inspection of traffic
- Send DNSSEC queries to DNSSEC servers
- Send abusive traffic to “abuse pool”
- Fix up case sensitive backends / clients
- Use regular expressions to route queries
- Client DoS worries: limit each host QPS or per /64
- Large scale DoS: absorb & filter at million QPS rates

Running **dnsdist**

No configuration

```
dnsdist -l 0.0.0.0:5300 8.8.8.8 208.67.222.222
```

- Listen on port 5300
- Serve on RFC 1918 addresses (default ACL)
- Distribute queries to Google and OpenDNS
- Use a sensible loadbalancing policy (leastOutstanding)

Simple configuration I

```
1  setLocal('192.168.1.92:53')
2  addACL('192.168.1.0/24') --  setACL would've taken
     ↪  out RFC1918
3  newServer{address='127.0.0.1:5300', qps=1000,
     ↪  order=1}
4  newServer{address='8.8.8.8', qps=10, order=2}
5  newServer{address='127.0.0.1:5301', order=3}
6  setServerPolicy(firstAvailable)
```

Simple configuration II

```
# dnsdist -C simple.lua
Added downstream server 127.0.0.1:5300
Added downstream server 8.8.8.8:53
Added downstream server 127.0.0.1:5301
Listening on 192.168.1.92:53
dnsdist 0.0.gf354a19 comes with ABSOLUTELY NO WARRANTY. This is free software, and you a
ACL allowing queries from: 127.0.0.0/8, 10.0.0.0/8, 100.64.0.0/10, 169.254.0.0/16, 192.1
Marking downstream 127.0.0.1:5300 as 'up'
Marking downstream 8.8.8.8:53 as 'up'
Marking downstream 127.0.0.1:5301 as 'down'
> showServers()
#   Name  Address         State  Qps  Qlim Ord Wt Queries Drops Drate Lat Outstanding
0         127.0.0.1:5300    up  0.0  1000   1  1       0     0   0.0  0.0           0
1         8.8.8.8:53        up  0.0    20   2  1       0     0   0.0  0.0           0
2         127.0.0.1:5301  down  0.0     0   3  1       0     0   0.0  0.0           0
All                             0.0                    0     0


> getServer(0):setDown()
> showServers()
#   Name  Address         State  Qps  Qlim Ord Wt Queries Drops Drate  Lat Outstanding
0         127.0.0.1:5300   DOWN  0.0  1000   1  1      18     0   0.0  9.4           0
1         8.8.8.8:53         up  0.0 10000   2  1       0     0   0.0  0.0           0
2         127.0.0.1:5301   down  0.0     0   3  1       0     0   0.0  0.0           0
All                             0.0                   18     0
```

Live traffic inspection I

```
> showResponseLatency()
Average response latency: 0.582 msec
   msec
   0.10 .
   0.20 ****
   0.40 ************************************************************
   0.80 ****
   1.60 .
   3.20
   6.40
  12.80
  25.60 *****
  51.20 ********
 102.40 *******
 204.80 *****
 409.60 ****
 819.20 *
1638.40 .
```

Live traffic inspection II

```
> topQueries(5)
   1  hehehey.ru.                                2358 23.6%
   2  localhost.                                 2281 22.8%
   3  time.apple.com.                             537  5.4%
   4  service-personal.de.                        144  1.4%
   5  time.euro.apple.com.                        109  1.1%
   6  Rest                                       4571 45.7%

> topSlow(4)
   1  148.117.189.193.in-addr.arpa.                 3  2.4%
   2  _sipfederationtls._tcp.helpdesk.symphony.com.my.    2  1.6%
   3  eu2-scloud-proxy.ssp.samsungosp.com.          2  1.6%
   4  219.116.189.193.in-addr.arpa.                 2  1.6%
   5  Rest                                        114 92.7%

> topResponses(2, dnsdist.SERVFAIL)
   1  150.209.45.194.in-addr.arpa.                 31 22.1%
   2  praesenzen.datevstadt.de.                    15 10.7%
   3  Rest                                         94 67.1%
```

Live traffic inspection III

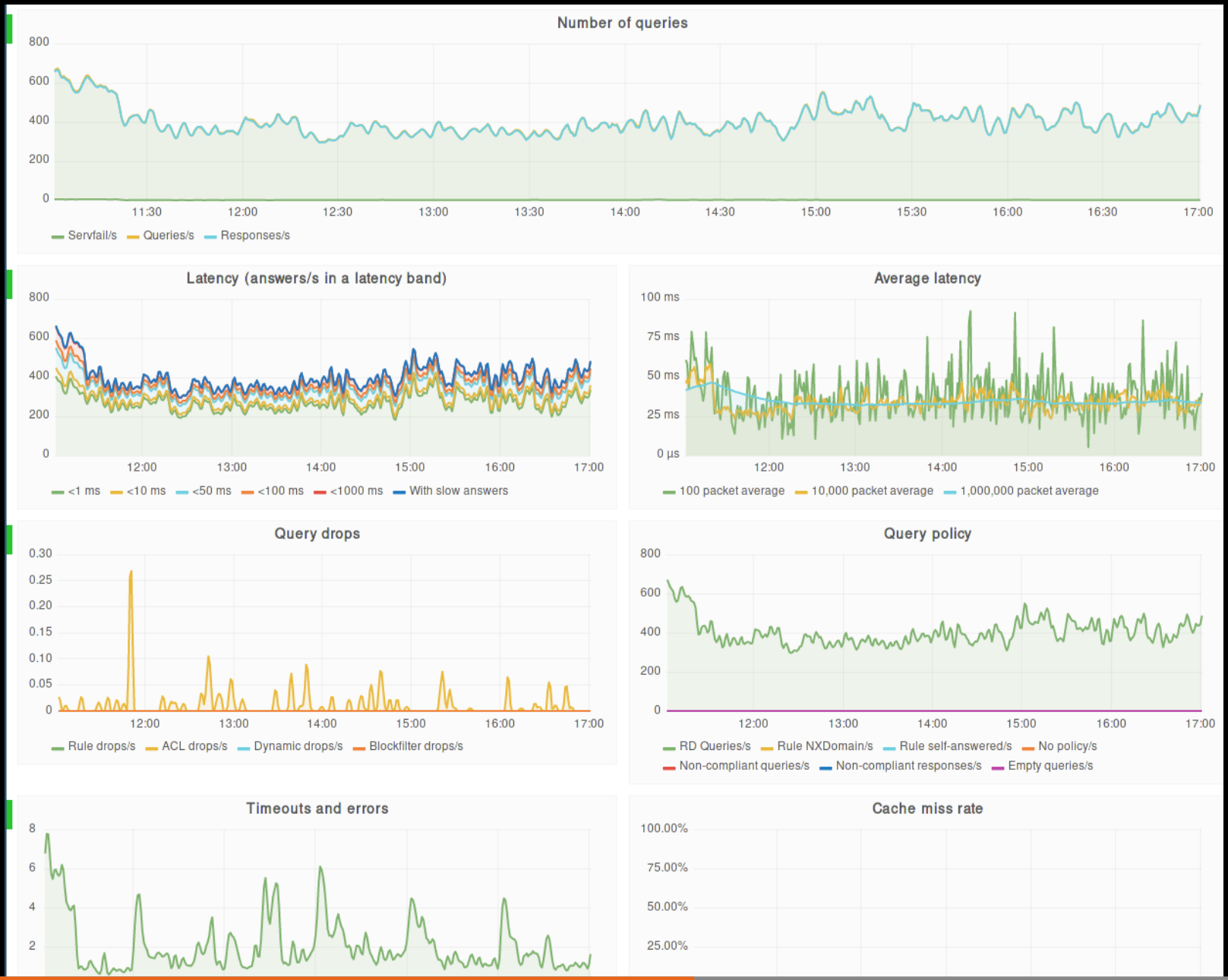
```
> grepq('ru', 2)
Time  Client              Server         ID    Name        Type Lat. TC RD AA Rcode
-0.2  192.168.1.92:33846                 4905  hehehey.ru. ANY          RD    Question
-0.2  192.168.1.92:33846  127.0.0.1:5300 4905  hehehey.ru. ANY  0.2     RD    Non-Existe
-0.2  192.168.1.92:33846                 4907  hehehey.ru. ANY          RD    Question
-0.2  192.168.1.92:33846  127.0.0.1:5300 4907  hehehey.ru. ANY  0.3     RD    Non-Existe

> grepq({'apple.com.', "100ms"}, 5)
Time    Client              Server         ID    Name           Type Lat.  TC RD AA Rcode
-127.6  192.168.1.92:43583 127.0.0.1:5300 44987 cl4.apple.com. A    247.2    RD    No Er
```

Statistics

```
1  carbonServer('37.252.122.50') -- send our statistics
     ↪  here
```

- Built-in export of metrics with the Carbon protocol
- Prometheus endpoint planned

Statistics


Number of queries
800
600
400
200
0
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00
16:30
17:00
Servfail/s
Queries/s
Responses/s
Latency (answers/s in a latency band)
800
600
400
200
0
12:00
13:00
14:00
15:00
16:00
17:00
<1 ms
<10 ms
<50 ms
<100 ms
<1000 ms
With slow answers
Average latency
100 ms
75 ms
50 ms
25 ms
0 µs
12:00
13:00
14:00
15:00
16:00
17:00
100 packet average
10,000 packet average
1,000,000 packet average
Query drops
0.30
0.25
0.20
0.15
0.10
0.05
0
12:00
13:00
14:00
15:00
16:00
17:00
Rule drops/s
ACL drops/s
Dynamic drops/s
Blockfilter drops/s
Query policy
800
600
400
200
0
12:00
13:00
14:00
15:00
16:00
17:00
RD Queries/s
Rule NXDomain/s
Rule self-answered/s
No policy/s
Non-compliant queries/s
Non-compliant responses/s
Empty queries/s
Timeouts and errors
8
6
4
2
Cache miss rate
100.00%
75.00%
50.00%
25.00%

Statistics

```
> dumpStats()
acl-drops                     0    latency0-1                14016359
block-filter                  0    latency1-10                  30963
cache-hits                20163123 latency10-50                 27356
cache-misses                 69187 latency100-1000              12795
cpu-sys-msec               1084278 latency50-100                 2740
cpu-user-msec              4291735 no-policy                        0
downstream-send-errors           2 noncompliant-queries             0
downstream-timeouts            579 noncompliant-responses           0
dyn-block-nmg-size               0 queries                    22443509
dyn-blocked                      0 rdqueries                  22443496
empty-queries                    0 real-memory-usage                0
fd-usage                         0 responses                  2279804
latency-avg100               281.3 rule-drop                        0
latency-avg1000              409.1 rule-nxdomain                    0
latency-avg10000             574.3 self-answered                    0
latency-avg1000000           736.9 servfail-responses             307
latency-slow                  1410 trunc-failures                   0
                                   uptime                     506617
```

Console and configuration I

- Connect to the console (`dnsdist -c`)
- Execute single commands (`dnsdist -e`)

```
1  controlSocket('0.0.0.0') -- for the console
2  setKey('MXNeLFWHUe4363BBKrY06cAsH8NWNb+Se2eXU5+Bb74=')
     ↪  -- for crypto
```

Console and configuration II

- Commands with a side-effect are stored
- Can be retrieved with `delta()`
- This output can directly be pasted into the config

```
> delta()
-- Wed Oct 26 2016 16:26:22 CEST
getServer(1):setDown()
-- Wed Oct 26 2016 16:26:40 CEST
getServer(1):setUp()
-- Wed Oct 26 2016 16:26:47 CEST
getServer(0):setDown()
-- Wed Oct 26 2016 16:41:11 CEST
controlSocket('0.0.0.0') -- for the console
-- Wed Oct 26 2016 16:41:19 CEST
setKey('MXNeLFWHUe4363BBKrY06cAsH8NWNb+Se2eXU5+Bb74=') -- for crypto
-- Wed Oct 26 2016 16:41:32 CEST
webserver('0.0.0.0:8083', 'geheim2') -- instant webserver
```

Deep-dive

Rules

Rules have Selectors with Actions

Selector: does this rule apply?
Actions: Do X if I match

Rules evaluated top-to-bottom, first match wins

Selectors

- Source address
- Query features (QNAME, QTYPE, Flags)
- Number of entries in a packet sections
- Number of labels in the name
- Regular Expression
- Supports And, Or and Not

Actions

- Drop
- Route to Pool
- Truncate (TC=1)
- Return SERVFAIL, NOTIMP, REFUSED
- Return custom answer
- Delay response by n milliseconds
- Remove flags before passing to backend
- Add originating IP address in an EDNS option
- Log query to TCP/IP host via Protobuf

Examples – Specialized Protection

```
addAction(MaxQPSIPRule(5, 24, 64), DropAction()) -- 5 QPS, grouped by /24 on IPv4 and by /64 on IPv6
addDomainBlock("ru.") -- Block all .ru domains
addDisableValidationRule({'servfail.nl', '1.0.0.0/8'}) -- Disable DNSSEC for this name and source-IP
addQPSLimit('evildomain.example.', 3) -- Limit queries to 3 per second for evildomain.example
```

Examples – Netmask groups

```
1  nmg = newNMG()
2  nmg:addMask('198.51.100.0/24')
3  nmg:addMask('203.0.113.0/24')
4
5  selector = AndRule{QTypeRule(dnsdist.AAAA),
     ↪  RegexRule('powerdns')} -- match QTYPE AAAA and
     ↪  QNAME containing powerdns
6  selector = AndRule{selector, NetmaskGroupRule(nmg)}
     ↪  -- Add the netmask group to the rule
7  addAction(selector, DelayAction(100)) -- Delay the
     ↪  answers to the above selector with 100 ms
```

Examples – Lua Rules

```lua
function authOrRec(dq)
  if(dq.dh:getRD() == false)
  then
    return DNSAction.Pool, "auth"
  end
  return DNSAction.Pool, "recursor"
end
addLuaAction(".", authOrRec)
```

Dynamic Rules

- If defined, every second the `maintenance()` function is called
- Provides access to ringbuffers and helpers
- Can create dynamic, automatically expiring blocks
- e.g. excessive queries, timeouts, SERVFAILs, NXDOMAINs
- Can dynamically switch traffic to other pools

Dynamic Rules

```
1  function maintenance()
2  addresses = exceedNXDOMAINS(100, 10) -- Get the
     ↪  addresses that had more than 100 NXDOMAINs in
     ↪  the last 10 seconds
3  addDynBlocks(addresses, "Exceeded NXDomain", 60) --
     ↪  Block the addresses for a minute
4  end
```

Demo time!

Packages at

https://repo.powerdns.com

Documentation at **http://dnsdist.org**

Thank you for your attention