BIND 9

(Part 3 - Load Balancing With DNSdist)

Carsten Strotmann and the ISC Team

Welcome

Welcome to part three of our BIND 9 webinar series

In this Webinar

- Installation and configuration
- Applications for dnsdist
- Aggregating metrics across a cluster
- Cache concentration
- Load balancing for authoritative
- Load balancing for resolver

What is dnsdist

- dnsdist is an open source DNS load balancer
  - Homepage: https://dnsdist.org
  - License: GPL Version 2
- Developed by PowerDNS.COM B.V
  - dnsdist is independent from the PowerDNS authoritative DNS server and DNS resolver (although some souce code is shared)
  - dnsdist works with standard compliant DNS server, such as BIND 9
  - dnsdist works with any standards-compliant DNS server, including BIND 9

dnsdist features (1)

- Receives DNS traffic and forwards DNS requests to downstream DNS resolver or authoritative DNS server
  - fail-over or load-balancing policies
- Response cache
- dnsdist can detect abuse and can rate-limit or block abusive sources
- DNS-over-TLS and DNS-over-HTTPS support
- DNScrypt support

dnsdist features (2)

- eBPF Socket Filtering (Linux)
- Simple but expressive and flexible configuration via Lua (embedded programming language)
- Dynamic reconfiguration
- Remote HTTP API
- Built-in web-server for API and statistics website

Installation and configuration

OS packages

- DNSDist is available in many Unix/Linux operating-system repositories
  - Debian/Ubuntu
  - Fedora
  - Red Hat EL / CentOS (via EPEL)
  - Arch Linux (AUR)
  - NixOS
  - FreeBSD / NetBSD / OpenBSD / DragonFlyBSD
  - pkgsrc (Cross-Platform https://www.pkgsrc.org)
- Distribution repositories might not have the latest release version available!

PowerDNS repositories

- PowerDNS.COM B.V. offers binary packages of the latest release versions plus the current development version
  - Debian 9/10
  - Raspbian/RaspberryOS 9/10
  - Ubuntu LTS 16.04/18.04/20.04
  - CentOS 7/8 (requires dependencies from EPEL)
- Information on these repositories can be found at https://repo.powerdns.com/
- PowerDNS.COM B.V. (part of Open Xchange https://www.open-xchange.com) offers commercial support for dnsdist

From Source

- dnsdist can be installed from source
- Dependencies
  - Boost
  - Lua 5.1+ or LuaJit
  - Editline (libedit)
  - libsodium (optional)
  - protobuf (optional, not needed as of 1.6.0)
  - re2 (optional)
- dnsdist (and other software) should **not** be compiled on a production machine
- Installation instructions can be found on https://dnsdist.org/install.html

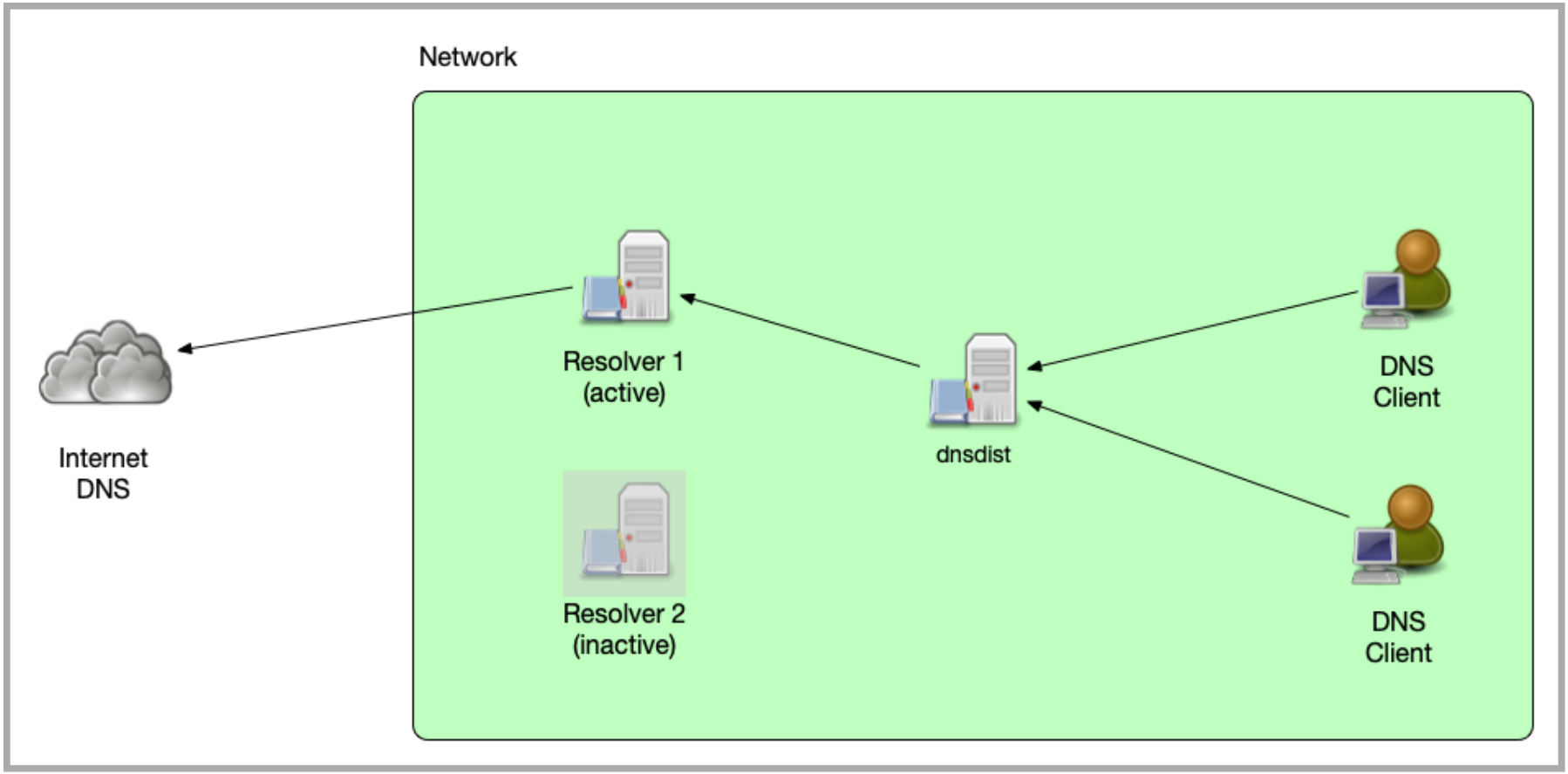
BIND 9

Applications of dnsdist

Fail-Over

- dnsdist can distribute queries among a pool of back-end servers based on the availability
  - Use the policy "firstAvailable"
  - The server in a pool have an **order**, the server with the lowest order being available will get all queries
  - This policy can be configured with an additional "queries per second (QPS) limit".
    - If the configured QPS limit of a server is reached, additional queries are spilled over to the next available server

Fail-Over



Load-Balancing

- dnsdist can distribute DNS queries across multiple back-end servers (or back-end server pools) based on several load-balancing policies:
  - `leastOutstanding`: use the server with the least outstanding queries (possibly least load)
  - `chashed`: distribute based on hashes of the query name (sticky queries)
  - `whashed`: distribute based on hashes of the query name (sticky queries), but apply the configured weight for the back-end server
  - `wrandom`: distribute random, but with a **weight** applied. Back-end server receive the share of queries based on their configured weight
  - `roundrobin`: distribute queries to all back-end server based on an round-robin algorithm (send each query to the next server)

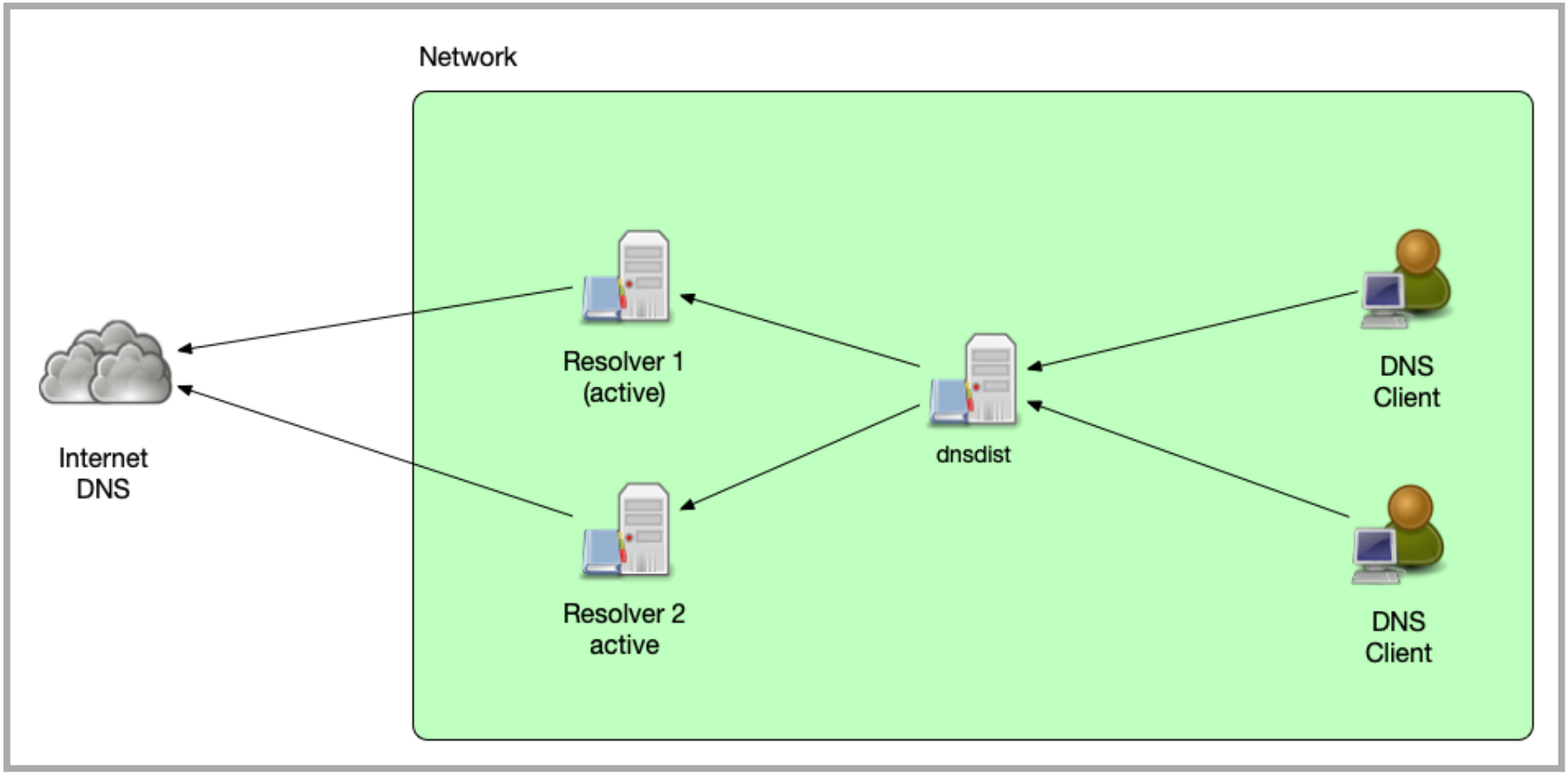
Load-Balancing

- dnsdist can be augmented with the Lua embedded programming language (https://www.lua.org/)
  - in addition to the built-in load-balancing policies, the administrator can add own policies written as small Lua snippets.
  - Example of a simple round-robin scheme:

```
counter=0
function luaroundrobin(servers, dq)
     counter=counter+1
     return servers[1+(counter % #servers)]
end

setServerPolicyLua("luaroundrobin", luaroundrobin)
```

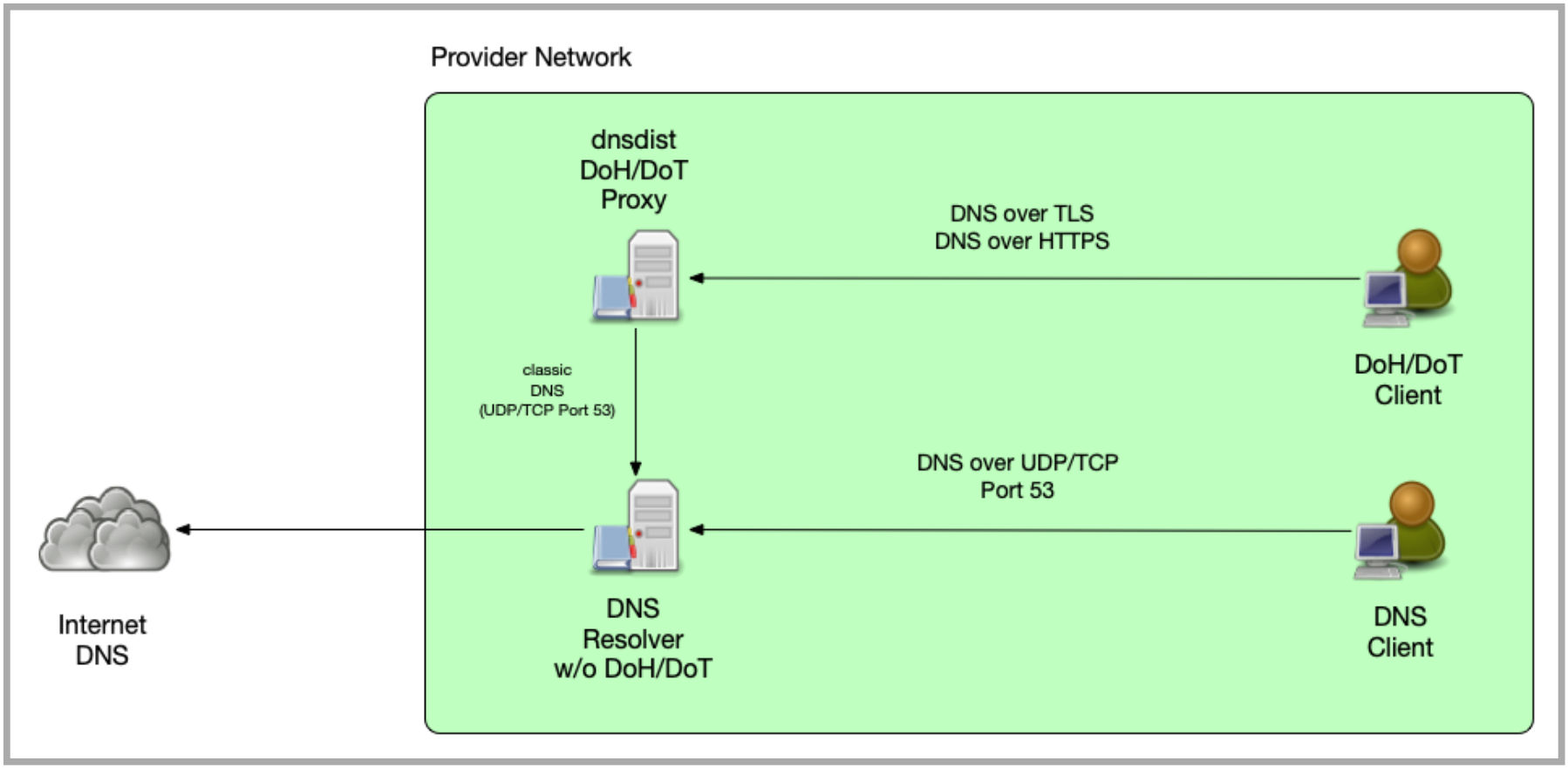
Load-Balancing



DoH/DoT Proxy and DDoS and Malware protection

- dnsdist can be used to add new DNS features to an existing DNS resolver or authoritative DNS server, without the need to make changes to the back-end server
  - Add DDoS protection
  - Add Malware Domain filtering
  - Add DNS-over-TLS or DNS-over-HTTPS

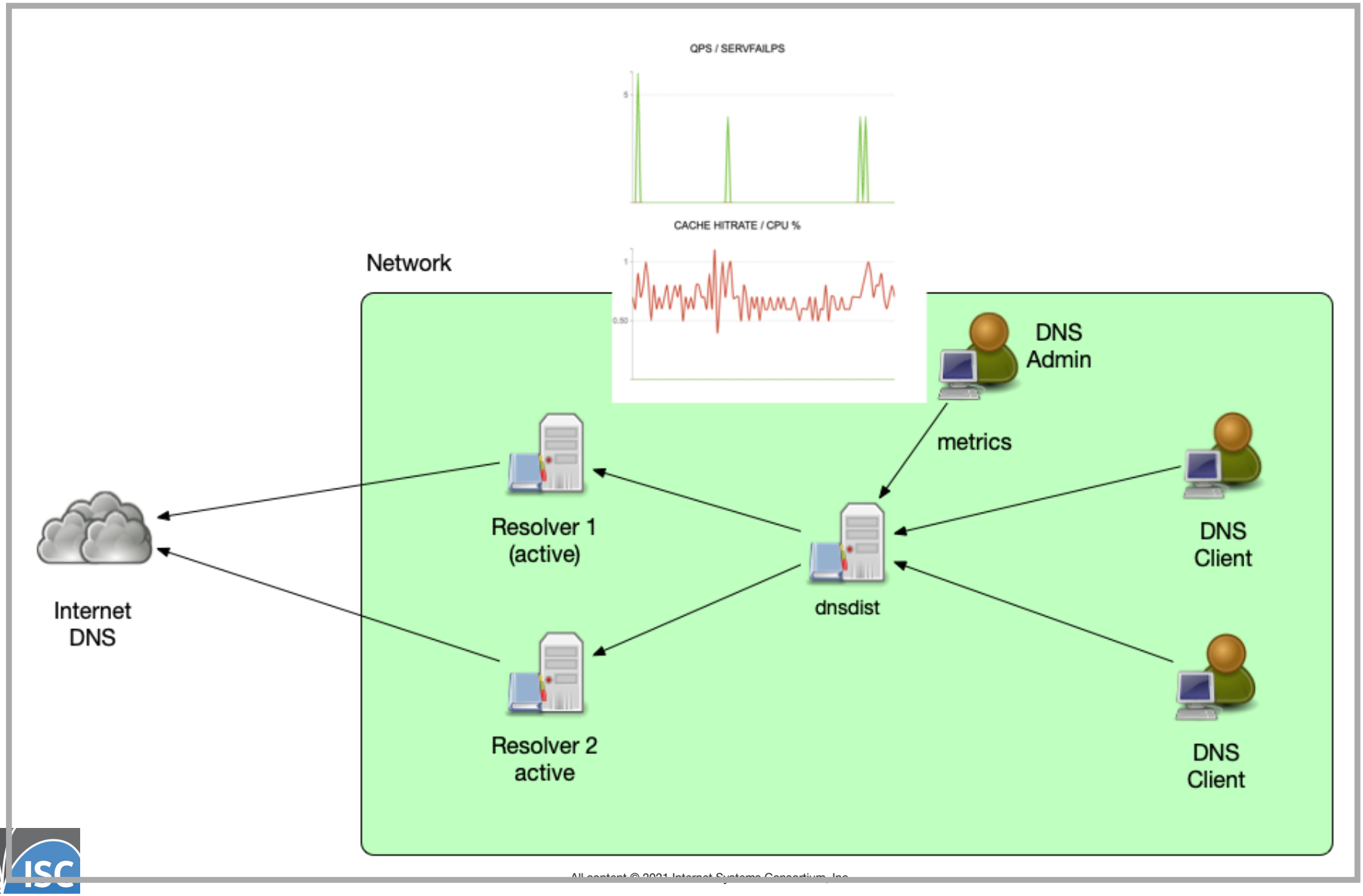
DoH/DoT Termination



Aggregating metrics across a cluster

- As dnsdist is the central system distributing DNS queries towards the back-end systems, it can be used to aggregate monitoring and metrics for a cluster of DNS machines
  - for multiple DNS resolvers
  - for multiple authoritative DNS servers

Aggregating metrics across a cluster

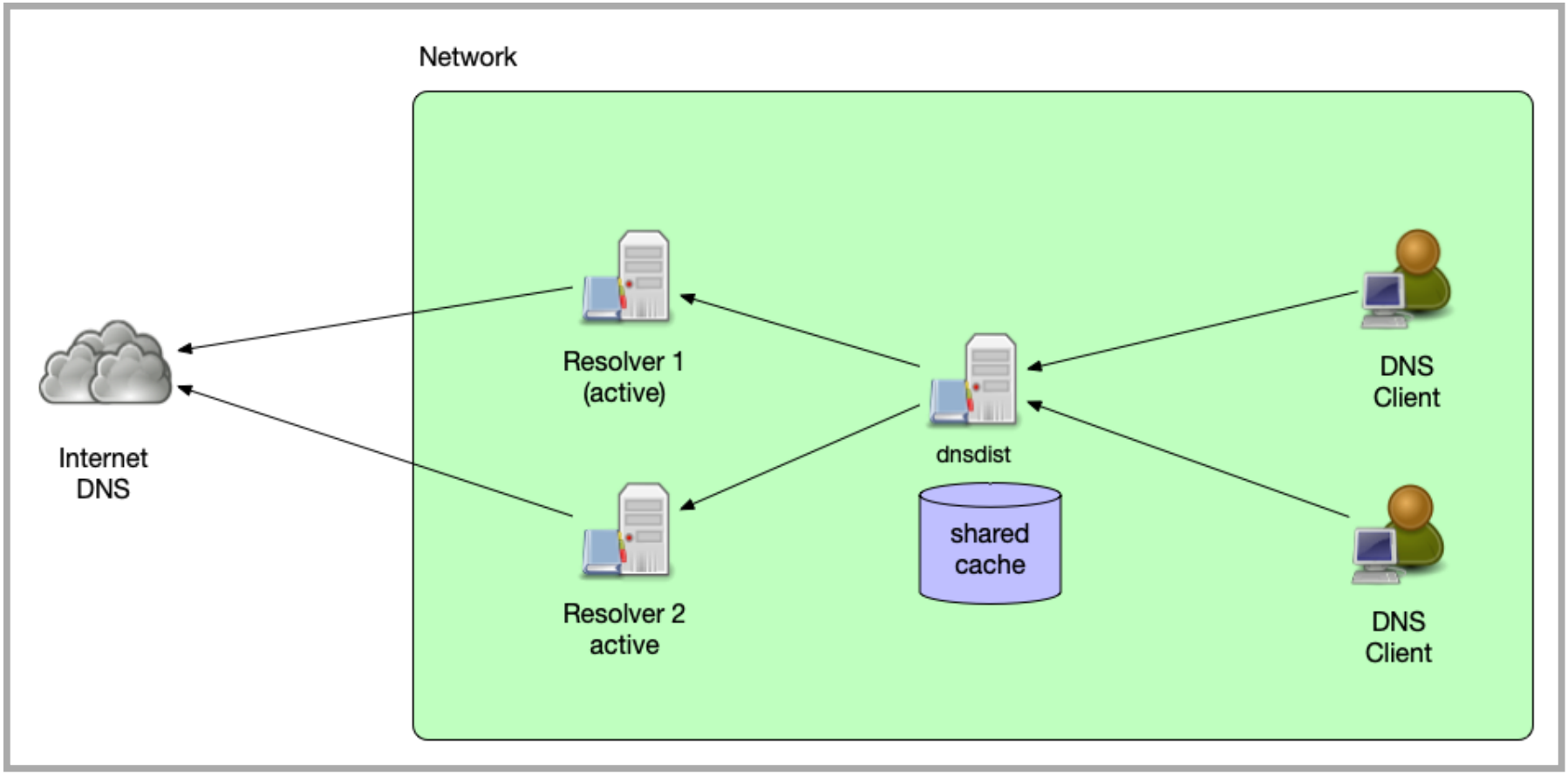


Cache concentration

- dnsdist is not a DNS resolver, it cannot follow delegations and resolve names
  - However dnsdist can cache response packets coming from downstream servers and can send responses to queries from the cache
  - dnsdist can be configured to serve *stale* (TTL expired) DNS data if no downstream server is available

```
> getPool(""):getCache():printStats()
Entries: 122/10000
Hits: 9147
Misses: 10147
Deferred inserts: 1
Deferred lookups: 0
Lookup Collisions: 0
Insert Collisions: 0
TTL Too Shorts: 0
```

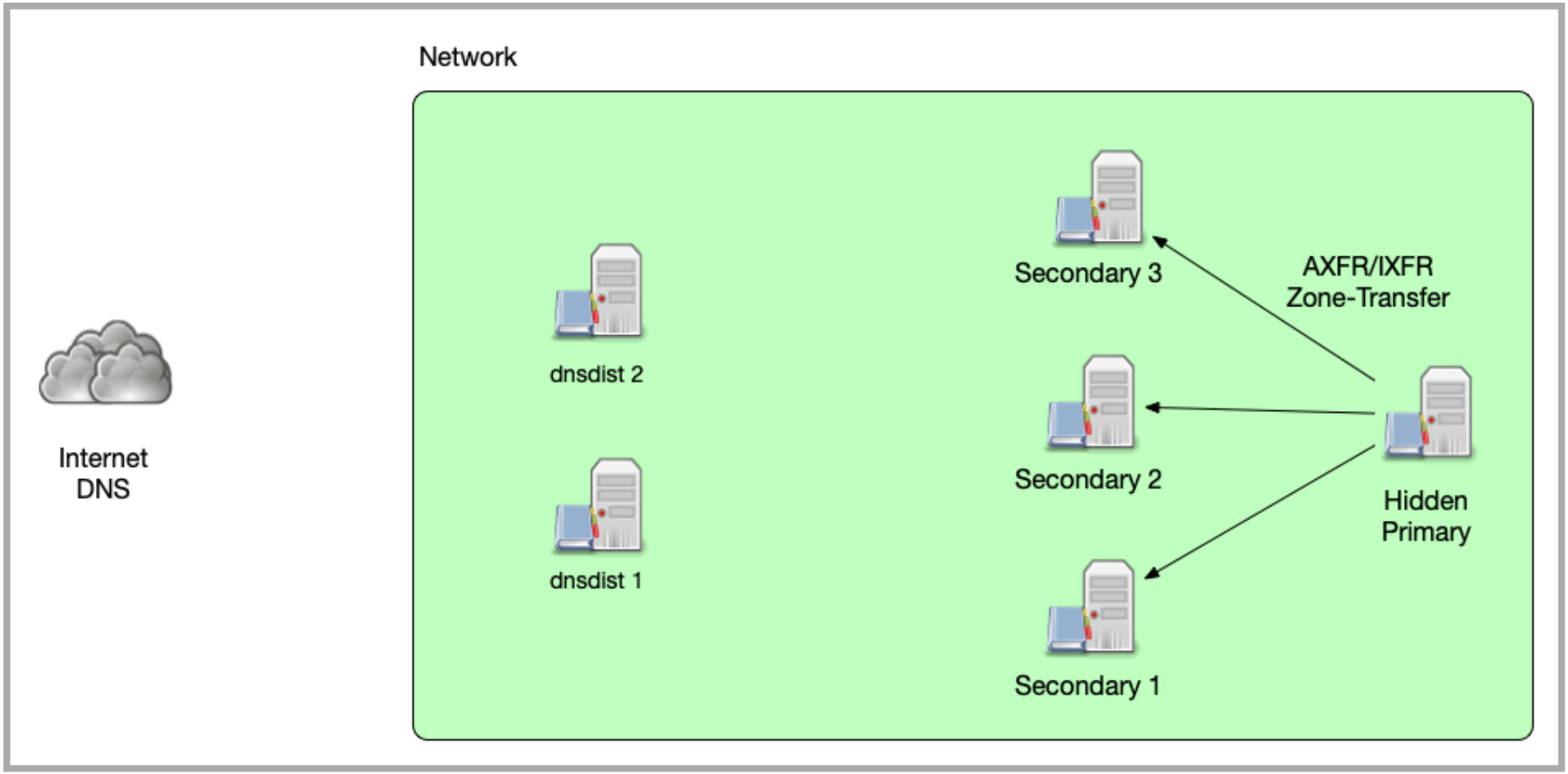
Cache concentration



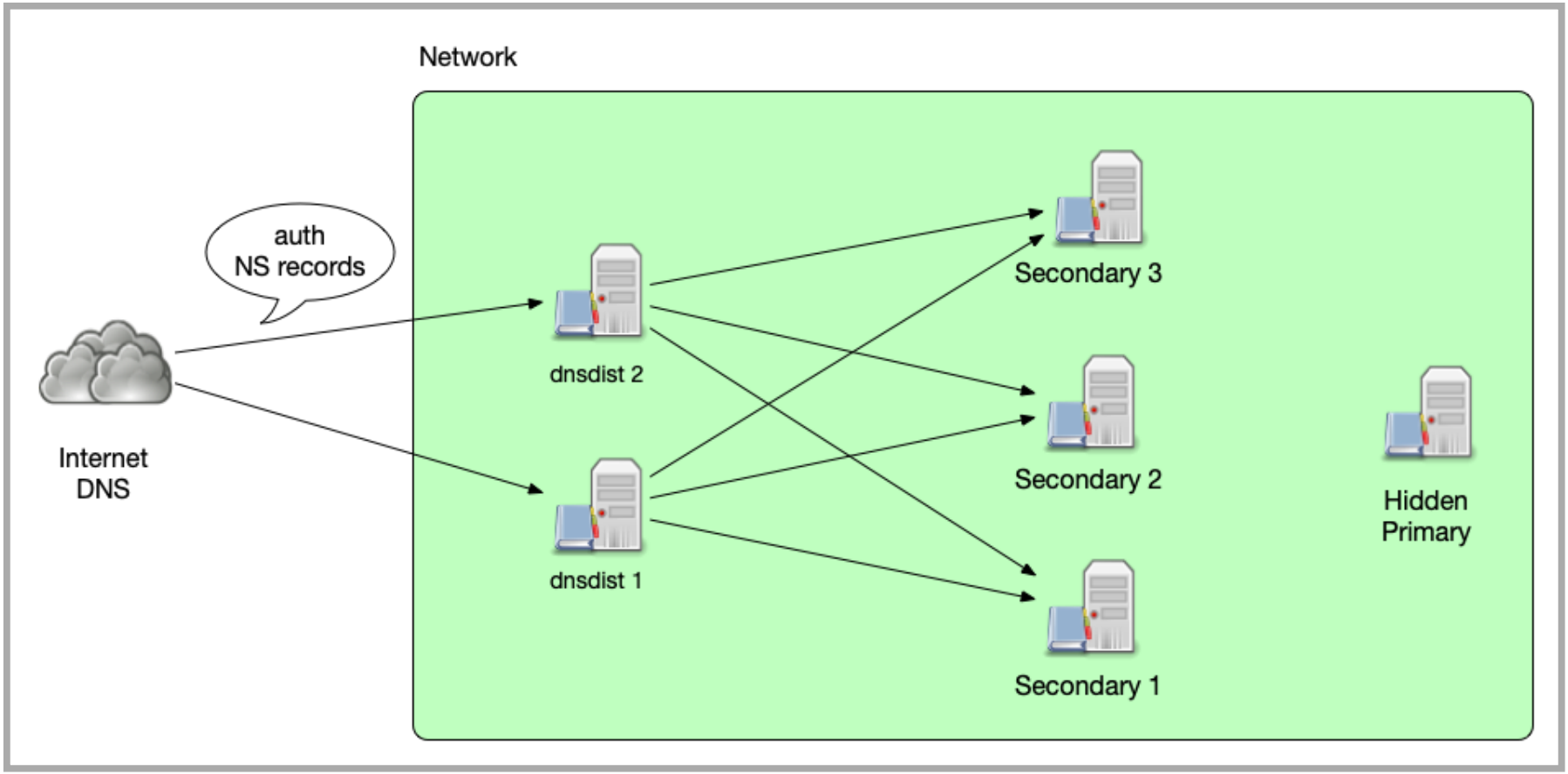
Auth-Server Resilience

- dnsdist can be a front-end load-balancer for authoritative server
  - Using the traffic rules dnsdist can guard the authoritative DNS server against some malicious traffic
  - Back-end authoritative servers can be taken offline without impact on service availability

Auth-Server Resilience



Auth-Server Resilience



Configuration and deployment

dnsdist high availability

- When using dnsdist it is important not to create a new single point of failure
- Possible solutions to make dnsdist highly available:
  - Configure multiple dnsdist instances via DHCP for DNS client traffic towards resolver
  - Configure multiple dnsdist instances via NS delegation records for resolver to authoritative traffic
  - Make a dnsdist instance high available through operating system clustering (Heartbeat/Pacemaker)

Configuration file

- dnsdist startup configuration is read from the file `dnsdist.conf` (usually in `/etc/dnsdist`)
  - this configuration file is a small Lua source file that is read and executed by the embedded Lua VM

Configuration file

- Example `dnsdist.conf`

```
---- Listen addresses
addLocal('192.0.2.1:53',     { reusePort=true })
addLocal('127.0.0.1:53',     { reusePort=true })
addLocal('[::1]:53',         { reusePort=true })
addLocal('[2001:db8::1]:53', { reusePort=true })
---- Back-end server
newServer({address="192.0.2.100",        qps=10000, order=1})
newServer({address="2001:db8:100::5353", qps=100,   order=3})
newServer({address="2001:db8:200::6312", qps=100,   order=2})
---- Policy
setServerPolicy(whashed)
setACL({'192.0.2.0/24', '2001:db8::/64'})
---- Cache
pc = newPacketCache(10000, {maxTTL=86400, minTTL=0, temporaryFailureTTL=60, staleTTL=60, dontAge=false})
getPool(""):setCache(pc)
---- Web-server
webserver("192.0.2.1:8083")
setWebserverConfig({acl="192.0.2.10/32",password="dnsdist-is-great"})
--- Console
controlSocket('127.0.0.1:5199')
setKey("2ux3QDmpdDAzYjspexaspAdqnXF8jXFU5qhd/BqXV8ag=")
---- Filter Rules
addAction(RegexRule(".*\\.facebook\\..*$"), RCodeAction(DNSRCode.REFUSED))
addAction(RegexRule(".*\\.doubleclick\\..*$"), RCodeAction(DNSRCode.REFUSED))
```

dnsdist console

- The dnsdist executable can connect as a remote CLI console to a running dnsdist
  - From inside this CLI console, it is possible to dynamically reconfigure dnsdist without restart

```
$ /bin/dnsdist -c
> showServers()
198.51.100.12#   Name                 Address                        State      Qps    Qlim Ord Wt    Que
0   192.0.2.53:53        192.0.2.53:53                  up     1.0   10000   1  1      10088     132
1   198.51.100.12:53     198.51.100.12:53               up     0.0     100   2  1       1391       2
2   203.0.113.11:53      203.0.113.11:53                up     0.0     100   3  1        318       0
All                                                             0.0                     11797     134
> newServer({address="1.1.1.1",        qps=10000, order=1})
1.1.1.1:53
> showServers()
#   Name                 Address                      State      Qps    Qlim Ord Wt    Queries   Drops
0   192.0.2.53:53        192.0.2.53:53                  up     0.0   10000   1  1      10103     132
1   1.1.1.1:53           1.1.1.1:53                     up     0.0   10000   1  1          3       0
2   198.51.100.12:53     198.51.100.12:53               up     0.0     100   2  1       1392       2
3   203.0.113.11:53      203.0.113.11:53                up     0.0     100   3  1        319       0
All                                                             0.0                     11817     134
>
```

dnsdist Web-server

- dnsdist can serve some internal metrics via an built-in web-server
  - this web-server needs to be configured in the configuration

```
---- Webserver
webserver("192.0.2.1:8083")
setWebserverConfig({acl="192.0.2.10/32",password="dnsdist-is-great"})
```

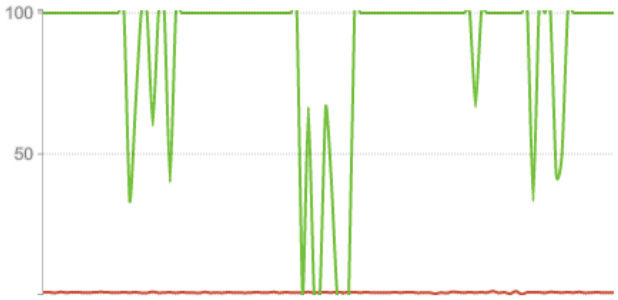
dnsdist Web-server

dnsdist 1.6.0-alpha3
dnsdist comes with ABSOLUTELY NO WARRANTY. This is free software, and you are welcome to redistribute it according to the terms of the GPL version 2.

Uptime: 4 hours, Number of queries: 22900 (1.00 qps), ACL drops: 0, Dynamic drops: 0, Rule drops: 0
Average response time: 6.79 ms, CPU Usage: 0.90%, Cache hitrate: 100.00%, Server selection policy: leastOutstanding
Listening on: 127.0.0.1:53, 172.22.1.8:53, [::1]:53, [fd75:8765:1d2a:0:a90a:6c20:75a4:d5dd]:53, ACL: 0.0.0.0/0, ::/0

QPS / SERVFAILPS

CACHE HITRATE / CPU %



#	Name	Address	Status	Latency	Queries	Drops	QPS	Out	Weight	Order	Pools
0	127.0.0.11:53	127.0.0.11:53	up	33.54	10186	140	0.00	0	1	1	
1	1.1.1.1:53	1.1.1.1:53	up	12.35	69	3	0.00	0	1	1	
2	172.22.1.1:53	172.22.1.1:53	up	49.72	1413	2	0.00	0	1	2	

#	Rule	Action	Matches
0	Regex: .*\.facebook\..*$	set rcode 5	0
1	Regex: .*\.doubleclick\..*$	set rcode 5	2

#	Response Rule	Action	Matches
No response rules defined			

Dyn blocked netmask	Seconds	Blocks	Reason
No dynamic blocks active			

Kernel-based dyn blocked netmask	Seconds	Blocks
No eBPF blocks active		

dnsdist Web-API

- Utilizing the web-server, dnsdist exposes a web API that can be used to
  - query statistics in JSON format
  - query metrics in Prometheus format
  - read the current running configuration from an dnsdist instance
- The web API access is authenticated by an API key that is sent with every API request

BIND 9

Aggregating metrics across a cluster

Graphite Monitoring

- Graphite is an open source, Python based monitoring application: https://graphiteapp.org/
- dnsdist can send metrics into an Graphite server using the native *carbon* protocol
- Example dnsdist configuration:

```
carbonServer('192.0.2.210', 'dnsdist.isp.example', 30, 'dnsdist', 'main')
```

- see https://dnsdist.org/guides/carbon.html

dnsdist and Prometheus

- Prometheus is a popular monitoring solution: https://prometheus.io
- Prometheus can read the dnsdist statistics from the `/metrics` URL endpoint of the Web-API

Cache concentration

- dnsdist can have one or more packet caches
  - caches can be separated by pool
- The caches hold the responses coming from back-end server (DNS resolver or authoritative)
- Example configuration:

```
pc = newPacketCache(10000, --- create a new pool cache "pc" with 10.000 entries
  {
  maxTTL=86400,            --- maximum TTL cache time
  minTTL=0,                --- minimum TTL cache time
  temporaryFailureTTL=60,  --- TTL used for server failures or "refused"
  staleTTL=60,             --- TTL for stale cache entries
  dontAge=false            --- cache entries "age", their TTL is decremented in cache
})
getPool(""):setCache(pc)   --- assign the cache to the default pool
```

Load balancing for authoritative DNS server

Health-check configuration

- By default, dnsdist sends the query for `a.root-servers.net` A-Record towards the downstream server
  - this will not succeed against most authoritative only servers
  - so the heath check needs to be adjusted. This example is forwarding towards the authoritative DNS servers for `isc.org`:

```
newServer({address="51.75.79.143",   checkType="SOA", checkType=DNSClass.IN, checkName="isc.org"})
newServer({address="199.6.1.52",     checkType="SOA", checkType=DNSClass.IN, checkName="isc.org"})
newServer({address="199.254.63.254", checkType="SOA", checkType=DNSClass.IN, checkName="isc.org"})
newServer({address="149.20.1.73",    checkType="SOA", checkType=DNSClass.IN, checkName="isc.org"})
setServerPolicy(leastOutstanding)
setLocal("192.0.2.123:53")
[...]
```

SOA queries and IXFR/AXFR

- SOA queries and IXFR/AXFR requests for zone transfer sync should be redirected by a dnsdist rule to a single authoritative downstream server
  - to make sure that the zone transfer is initiated from the same zone data-set that was seen in the SOA query
  - the following configuration snippet sends all SOA/AXFR and IXFR requests towards the pool `primary`, which only contains one primary authoritative server

```
newServer({
  address="192.0.2.123",
  name="primary",
  pool={"primary", "otherpool"}
})
addAction(
  OrRule({
    QTypeRule(DNSQType.SOA),
    QTypeRule(DNSQType.AXFR),
    QTypeRule(DNSQType.IXFR)}),
  PoolAction("primary")
)
```

SOA queries and IXFR/AXFR

- The back-end authoritative DNS servers will see the requests (SOA, Zone-Transfer) coming from the dnsdist IP-Address(es)
  - Access Control Lists on the back-end server must be adjusted accordingly
  - The `source` parameter tells dnsdist which IP-address or interface to use for outgoing queries:

```
newServer({address="192.0.2.1", source="192.0.2.127"})
newServer({address="192.0.2.1", source="eth1"})
newServer({address="192.0.2.1", source="192.0.2.127@eth1"})
```

Dynamic Updates

- DNS dynamic updates (RFC 2136) should be sent to a real primary authoritative DNS server, not towards dnsdist
  - This can be done with the name of a real authoritative DNS server in the SOA records `mname` field
  - Or by manually instructing the dynamic DNS client (like `nsupdate`) to use a dedicated IP address:

```
nsupdate
> ttl 3600
> server 192.0.2.221
> add www.example.com. IN A 192.0.2.212
> send
```

Notify

- An updated authoritative DNS server will sent `notify` messages to all secondaries configured in the NS records
  - In case of a deployment with dnsdist, this might be a dnsdist instance that will forward the notify towards the back-end server(s)
  - IP based ACLs in use at the back-end server need to be adjusted to include the dnsdist source address
  - ACLs in dnsdist can take over the role of the ACL of the authoritative server, only allowing `notify` from trusted source addresses
  - TSIG based ACLs have no issue, as they are independent of the IP addresses used
  - As an alternative, `notify` can be configured explicitly on the authoritative servers. Example for BIND 9:

```
zone "example.com" {
   type primary;
   file "example.com";
   notify explicit;
   also-notify { 192.0.2.53; 198.51.100.12; };
};
```

Rate-Limiting

- dnsdist can filter or rate-limit DNS traffic based on matching packets (selectors)
  - DNSSEC or not
  - EDNS option
  - Max QPS per IP/Subnet
  - Source Network
  - DNS Opcode
  - DNS network class
  - Query Name (Regular Expression)
  - Number of labels in the query name
  - Return Code
  - RD-Flag (Recursion Desired)
  - Number of Records / Number of types of record in response
  - and many more

Rate-Limiting

- Rules in dnsdist can be dynamically inserted based on observed traffic (dynamic rules)
  - Through the Lua programming language, the rule decisions can be adjusted to the operator's need
- Rules can automatically "age out" after some time to prevent over-blocking

```
local dbr = dynBlockRulesGroup()
dbr:setQueryRate(30, 10, "Exceeded query rate", 60)
dbr:setRCodeRate(DNSRCode.NXDOMAIN, 20, 10, "Exceeded NXD rate", 60)
dbr:setRCodeRate(DNSRCode.SERVFAIL, 20, 10, "Exceeded ServFail rate", 60)
dbr:setQTypeRate(DNSQType.ANY, 5, 10, "Exceeded ANY rate", 60)
dbr:setResponseByteRate(10000, 10, "Exceeded resp BW rate", 60)

function maintenance()
  dbr:apply()
end
```

- Dynamic Rules: https://dnsdist.org/guides/dynblocks.html

Rate-Limiting

- Using the built-in Rules, many types of malicious traffic can be blocked or redirected
  - On Linux, with the help of eBPF, dnsdist can block certain DNS traffic when entering the Kernel without going through the whole TCP/IP stack
    - this can reduce the load in case of an DDoS attack
- eBPF Socket Filtering: https://dnsdist.org/advanced/ebpf.html

BIND 9

Load balancing for resolver

Fail-over Configuration

- Setting the dnsdist load-balancing policy to `firstAvailable` will create a simple fail-over configuration
  - all queries go to the first available server in the configured order that have not exceeded its configured QPS (queries per second) limit

```
---- Back-end server
newServer({address="192.0.2.100",        qps=1000, order=1})
newServer({address="2001:db8:100::5353", qps=500,  order=2})
newServer({address="2001:db8:200::6312", qps=500,  order=3})
---- Policy
setServerPolicy(firstAvailable)
setACL({'192.0.2.0/24', '2001:db8::/64'})
---- Cache
pc = newPacketCache(10000, {maxTTL=86400, minTTL=0, temporaryFailureTTL=60, staleTTL=60, dontAge=false}
getPool(""):setCache(pc)
[...]
```

Load-Balancing Configuration

- The other available server policy options in dnsdist create load-balancing configurations:

```
---- Back-end server
newServer({address="192.0.2.100",        order=1})
newServer({address="2001:db8:100::5353", order=3})
newServer({address="2001:db8:200::6312", order=2})
---- Policy
setServerPolicy(leastOutstanding)
setACL({'192.0.2.0/24', '2001:db8::/64'})
---- Cache
pc = newPacketCache(10000, {maxTTL=86400, minTTL=0, temporaryFailureTTL=60, staleTTL=60, dontAge=false})
getPool(""):setCache(pc)
[...]
```

- Custom policies can be written in the embedded Lua programming language
- Loadbalancing and Server Policies https://dnsdist.org/guides/serverselection.html

Server Pools

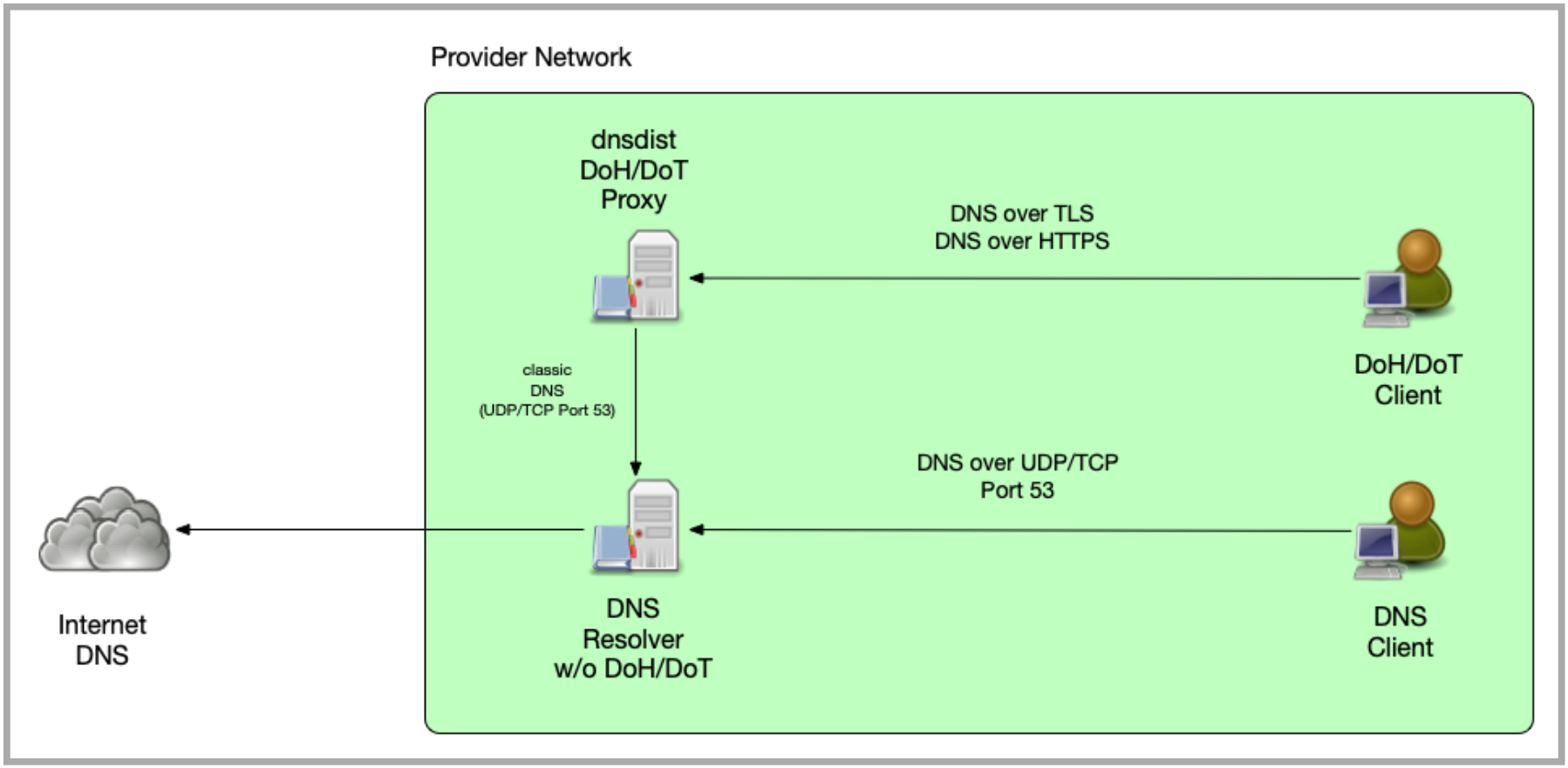
- dnsdist groups back-end servers by "pools"
  - there is always the default pool with the empty name ""
  - additional pools can be created when adding new back-end server
  - Rules and Actions can be used to select the pool for certain queries
- Pools can be used to isolate *bad* queries (DDoS, Malware)
  - excessive query rates or problematic queries from malware infected clients can be isolated so that regular users are not effected

```
-- Add a backend server with address 192.0.2.3 and assign it to the "abuse" pool
newServer({address="192.0.2.3", pool="abuse"})
-- Send all queries for "bad-domain1.example." and "bad-domain2.example" to the "abuse" pool
addAction({'bad-domain1.example', 'bad-domain2.example.'}, PoolAction("abuse"))
```

DoH/DoT Termination

- dnsdist can be used to terminate DNS-over-TLS (DoT) and DNS-over-HTTPS (DoH) traffic
  - it creates a DoH/DoT "proxy" that receives DoH/DoT from client machines and forwards classic DNS over UDP/TCP Port 53 towards the back-end resolver (which could be BIND 9)

DoH/DoT Proxy



Why a DoH/DoT proxy?

- easy deployment
- existing DNS resolver infrastructure does not need to be touched
- scaling through separate hardware/server instances

Upcoming Webinars

- May 19: Session 4. Dynamic zones, pt1 - Basics
- June 16: Session 5. Dynamic zones, pt2 - Advanced topics

Questions and Answers