Managing sleep with a resilient infrastructure

DNS Recursive and Authoritive and how to deploy them all: introducing Ramc

QUICKLINE

Jeroen Massar <jeroen.massar@qlgroup.ch>

How to keep (the ops team) sleeping…

- By knowing what is running:
  - Deploy & Verify & making sure it is secure:
    - handle updates, especially security ones timely
    - manage firewall rules (open ports when needed, keep things closed)
    - restrict access: not everybody needs root, let alone even shell access to the host

How to keep (the ops team) sleeping…

- By knowing it is running ’well’: monitoring
  - SNMP / IPFIX / Netflow: collect, but collect what you need not ‘everything’.
    - Thus no overmonitoring: ensure that you do not have multiple hosts doing SNMP against your CMTS and overloading the poor little snmpd…
  - Have something / body look at it and file tickets and then somebody resolving tickets.

How to keep (the ops team) sleeping…

- By having redundancy:
  - double datacenter (L1)
  - double switches (L2)
  - double routers (L3)
  - double VM host (~L1)
  - double VM guest (~L3)
  - double software variants

Yep, uses bit more resources, but those are lightly loaded and typically cheaper than multiple engineers coming out of their bed.

Recursors

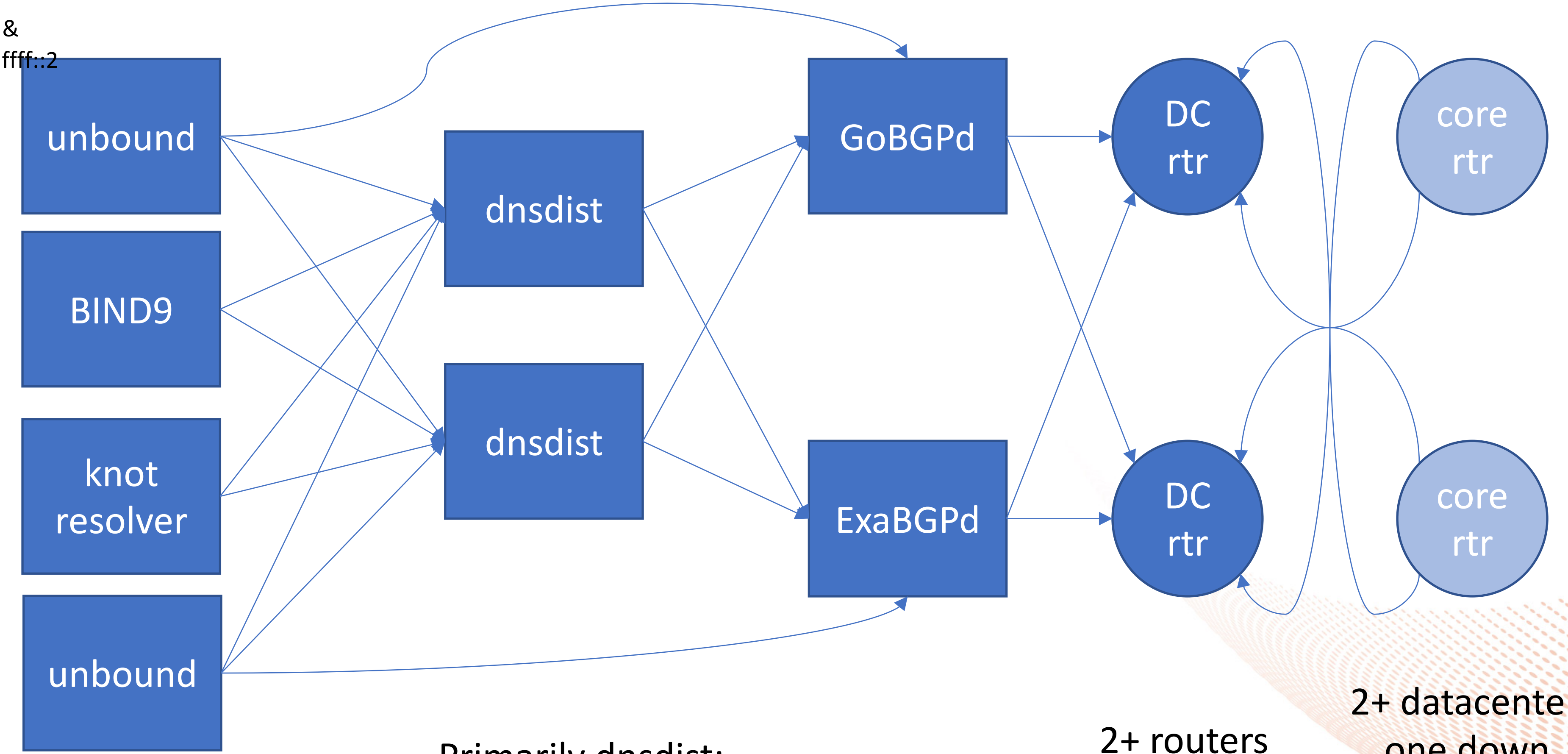
What Quickline customers use
(212.60.61.246 + 212.60.63.246 &
2001:1a88:10:ffff::1 + 2001:1a88:10:ffff::2

Variety of actual recursors; unbound fastest*, thus primary, partial amount of queries get balanced to BIND9 & knot, as backup and for diversity

*=in our testing YMMV

2 different BGP daemons (just in case of implementation issues)

2 Anycasted IPs (2x IPv4 + 2x IPv6)
One 'primary' for that BGP, the other 'backup'.



Primarily dnsdist; failover to unbound

2+ routers per DC

2+ datacenters one down, other one lives

Authoritive

What is used for domain names

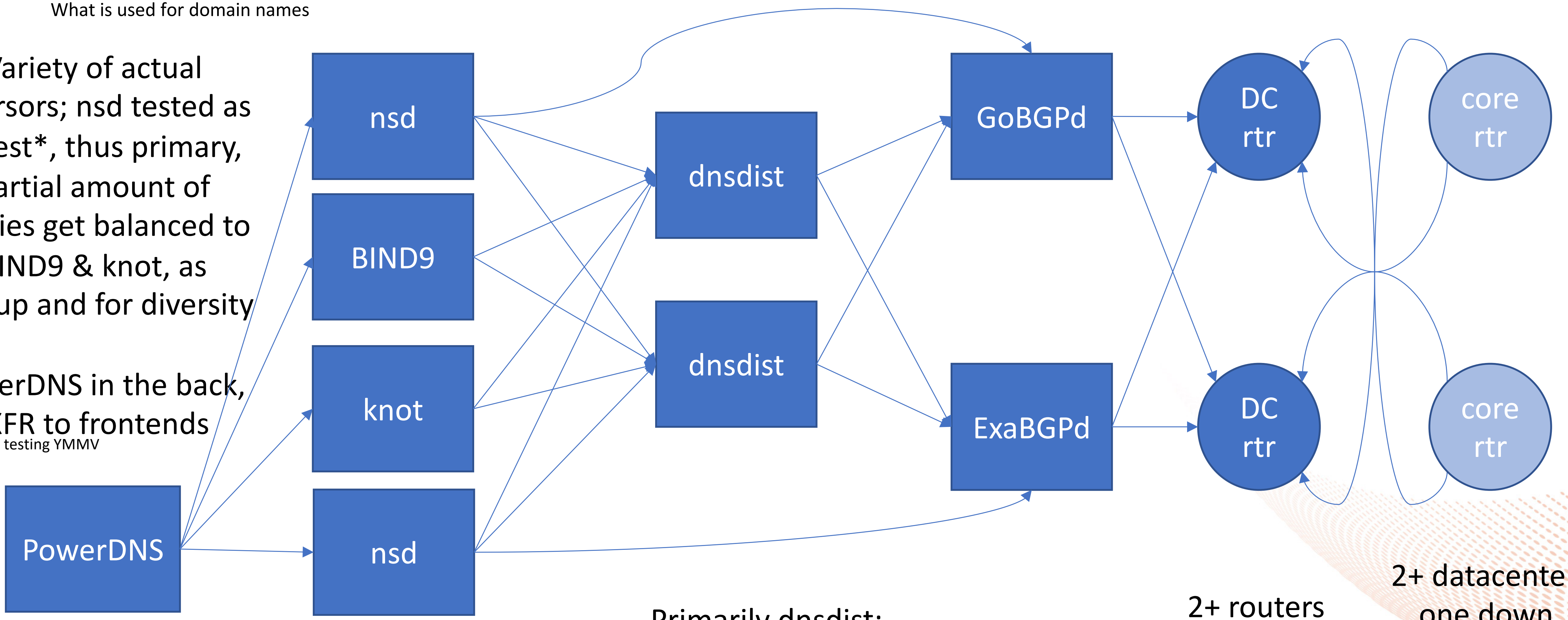
2 different BGP daemons (just in case of implementation issues)

3 Anycasted IPs (3x IPv4 + 3x IPv6)
One 'primary' for that BGP, the other 'backup'. (3rd site is primary for 3rd IP)

Variety of actual recursors; nsd tested as fastest*, thus primary, partial amount of queries get balanced to BIND9 & knot, as backup and for diversity

PowerDNS in the back, IXFR to frontends

*=in our testing YMMV



Primarily dnsdist; failover to nsd

2+ routers per DC

2+ datacenters one down, other one lives

So, who set that all up?

- Alex Jueni & Jeroen Massar did the design & configs
- The QL Netops Team and the Sysops Team normally deploys things
  - but not by hand…. anymore….

Ramc

- **R**emote **A**dministration, **M**anagement and **C**onfiguration
  pronounced: ramsey

- Not completely happy with Ansible:
  - Can’t see what was changed on the server (in every organisation somewhere sometime someone logs in as root and just installs things and/or ‘quickly’ modifies something and then it gets forgotten…. till that they you redeploy and that fix is thus mising)
- Other tools (puppet etc) do not match our requirements either or do not provide the integration we want
  - Want SSH based management with limited account where possible
  - Do not want another agent running on the host
  - Want integration with IP management, VLAN / VRF management & … DNS ☺

Ramc: steps of what it does

ramc follows the following steps (unless only specific steps are requested by parameters).

1. Once: Talk to VM Host to setup VM based on reservations and variables
2. Fetch live to local dir, compare with what was previously there
   - detects server-side changes (e.g. Debian upgrade or manual changes that did not come from us)
   - user can now commit these changes "accept" or "revert" (we overwrite them)
3. Make changes to server
   - update packages
4. Fetch live files to local dir, check changes
5. Apply file changes to 'local live'
6. Copy files to 'live server'
7. Restart services that need to
8. Verify what ports are listening and if something new/unexpected is open (netstat -anp compare)

Git all the configs!

- Keep configuration (recipes) in Git:
  - know what changed, when and by whom
- Let a tool push out configs to the devices (be that routers or servers, physical hardware or virtual machines)
- Store/backup your router config (dumps) in git, so one can compare them (read: new variant of rancid)
- Before pushing new config, verify that nobody changed the server
  - Can also do this routinely to see if an adversary changed something and thus detect these issues (not all adversaries are internal ;))

RAMC: A language for configuration

- RAMC is written in Go, but sysadmins/sysops/netops etc are typically scripters, not true programmers
- We have thus defined a simple ‘language’ that can use ‘includes’ to include snippets from other pieces and thus easily ‘built upon’ other snippets. (Docker does a similar thing, but misses many features)

test.example.com.ramc

```
# Variables
var host.name test.example.com
var host.ipv4_addr 192.0.2.2
var host.ipv6_addr 2001:db8::2

# Includes
include roles/network_standard
include roles/default
include roles/ntp_client
include roles/nginx
```

Package installs & configuration

```
# Install the nginx package
pkg install nginx

# Copy a file (the destination is the 'live'
directory for that host)
file copy -R --chown root:root files/ /
```

File Templates

- The file.* commands understand that if a filename ends in *.tmpl that that is a Golang based template (and thus can include code).
- The template is applied before copying, replacing values and calling functions where needed.
- After the copy and execution of the template the .tmpl extension is removed.
- One can also avoid executing a template by providing the option --verbatim.

Host Reservations

```
# reserves a few CPU cores (of undefined speed, might add a Ghz setting later)
reserve cpu 4

# reserves some memory (on top of minimal OS mem)
reserve memory 4G

# indicates how much space the application needs (on top of OS) repeat (inside includes) adds to that
reserve disk 20G

# indicates expected network traffic in (M) Megabits/sec
reserve network 100M

# exposes ports to the network, from this firewall rules are build (Docker has a similar thing!)
port expose 80 443
```

What Ramc does for us

- Provisions IP & DNS database
  - Can request/reserve an IP address for a given VLAN/VRF
  - Can register hostnames, request SSL certificates with that
- Talks to Vmware or KVM to setup a host in the requested VLAN / VRF
  - Permission model only allows certain groups of people to only request certain prefixes
- Generates iptables rules for the host

  (optionally for the KVM, so that firewalling is one layer above and cannot be modified by an adversary)

Upgrades with RAMC

- Re-run the RAMC script
- A ‘apt-get update && apt-get dist-upgrade’ is part of the steps RAMC takes, thus upgrading software
  - Yes, ‘dist’ upgrade: if Debian considers stable stable, then run that, don’t wait years before switching over
  - RAMC does the upgrade and then checks what Debian modified, asks upgrader if those changes are acceptable and stores the new configs in Git

Status of RAMC

- CLI only tool (no fancy UI yet; but go, thus easy to add when needed)
- Still in development / ~experimental stages
- Used for deploying a few platforms, more to follow, lots still to do
  - DNS Auth & Recursive, Speedtest (try our new one!), etc...
  - Currently servers & services, but routers and other network devices planned too
- Open Sourcing to follow (maybe a follow-up presentation at next SwiNOG? Will announce on the list)

How to move ”ancient” machines

- There is always a machine that was installed over the last ~decade where one does not know what really is on it but that is ‘operational’
- If luck, one has Debian (or Ubuntu etc) and thus: debsums
    - Debsums can tell what changed on a file.
- If luck, with Debian, one has package file lists.
    - One can thus determine which files are installed ‘by the system’ (Debian package) and which files thus are manually added
- Combo of debsums + package file lists: we know what is ‘different’ from the packaged files.
- Then gather the list of packages that are needed and the configuration deltas
- => We have a partially functioning implementation for gathering the package list generation (that excludes dependent packages and leaves the actually installed packages) and that filters out all known files, thus leaving the ‘unknown’ files.

Questions?

(for funny slides, wait for Pascal’s presentation…. ;))