Fault Aware Global Server Load Balancer in DNS

Stefan Caunter
Allan Jude

Target Audience

- Does your organization have more than one Point of Presence?
- Would you or your app benefit from geographically segmenting your traffic?
- Could you use the ability to deploy simple failover (active/passive) or multi-node load balancing?
- Do you want your server monitoring to automatically update your DNS?
- gDNSd (topic of this talk) was originally built at Logitech to manage directing users to nearby driver download mirrors

Us (ScaleEngine guys)

Allan Jude

- 10 Years as FreeBSD Server Admin
- Architect of the ScaleEngine ESC/CDN
- Professor - Mohawk College (2008-2011)
- Host of TechSNAP.tv Podcast

Stefan Caunter

- 12 Years as a *BSD Server Admin
- Web Ops at TheStar.com (Large Toronto Newspaper)
- Mohawk College / Seneca (2001 - 2009)

ScaleEngine runs on this open source GSLB

Overview

- Introduction to ScaleEngine
- Challenges with growth
- What is a Global Server Load Balancer?
  - What are the current solutions?
- gDNSd implementation
  - Defining a Response Policy
  - Advanced Response Policies with GeoIP
  - Use Cases and Example Configurations
  - Advanced Performance Agents
  - Active Performance Monitoring
  - On-Demand Capacity via DNS
  - EDNS-Client-Subnet Implementation
  - Working around the EDNS whitelist
- Lessons Learned

What is a ScaleEngine

ScaleEngine is a global CDN, Video Streaming provider and application hosting platform entirely powered by FreeBSD

- ESC - Edge Caching for small web objects
- CDN - Global Caching for all web objects
- VSN - Live and On-Demand Video Streaming to Desktop and Mobile (RTMP, RTSP, HLS)
- OWC - Accelerated PHP/MySQL application hosting
- GSLB - Managed Global Server Load Balancer

Under the hood

- 70 non-virtualized hosts
  - 25 different data centers
  - 9 different countries
  - Aggregate 1500GB of ram
  - Aggregate 50 gigabits/sec uplink capacity
- All running FreeBSD 9.x
- Managed by Puppet
- Extensive use of Jails (w/ ezjail)
- ESC powered by Varnish
- CDN/OWC powered by NGINX
- OWC uses cluster of jails running php-fpm

Stats

- The last year (Oct 1 2011 - Sept 30 2012)
  - Origin: 1451 M requests, 31.7 TB
  - CDN: 2.2 B requests, 66.0 TB
  - ESC: 34.2 B requests, 173.4 TB
  - VSN: 126 M requests, 1644 TB
- Sept 2012:
  - Origin: 185 M requests, 7.2 TB
  - CDN: 275 M requests , 9.5 TB
  - ESC: 5.3 B requests, 25.8 TB
    - Average: 2050 requests/sec
    - Peak: 4950 requests/sec
  - VSN: 17.8 M requests, 190.8TB
    - Peak: 17.9 gbps

We push a fair amount of bits

VIDEO: We have very spiky "event" driven traffic, from < 100 to 10000 megabits in 5 minutes

- Low request rate, extremely high bandwidth
- Long lived sessions
- Peak load

HTTP: Compared to very smooth, "constant demand" traffic from the http CDN

- High request rate, lower bandwidth
- Base load

Both benefit from geographic distribution, but for different reasons

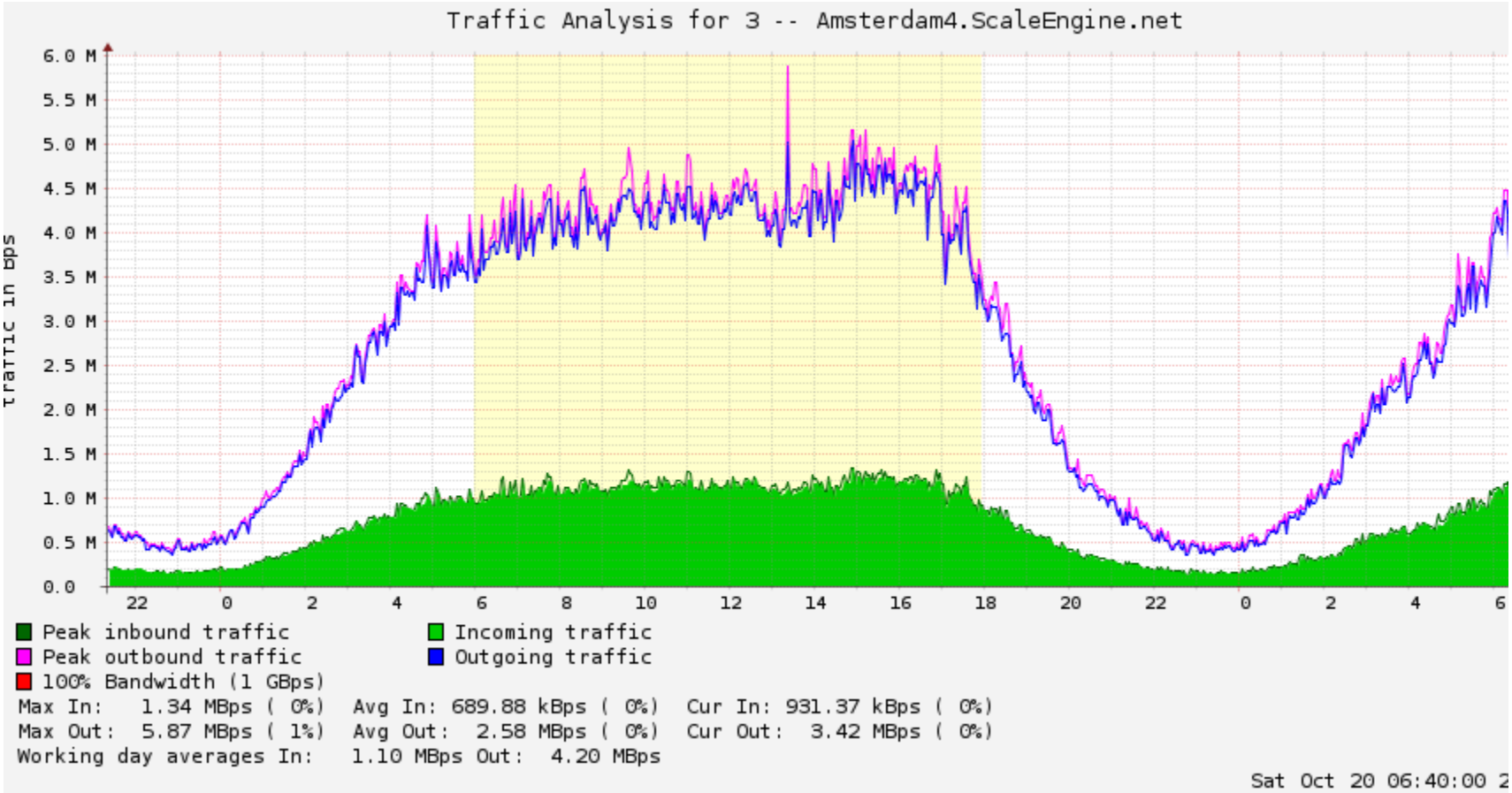
Some Background

- Our CDN got started after a large hosting customer quadrupled in size
- We were on a fixed commit with our expensive North American hosting provider, a modest 10mpbs with burst to 100mbps
- We found that getting additional servers from a cheaper provider, and creating a subdomain for image content offloaded enough traffic to avoid expensive overage charges on our permium transit

HTTP and subdomains

- Modern browsers will parallel download if you serve content from additional domains
- improves speed if user is reasonably close to server
- *does not improve speed if user is not reasonably close to server
- Separating domains means no cookies to block caching
- allows to cache popular content to reduce outbound traffic from origin

smooth HTTP graph



The Challenge

- It immediately becomes apparent that just creating a couple of CNAMEs for a subdomain is suboptimal and cumbersome to manage on a large scale
- Users do not benefit from any geographic awareness with round-robin DNS
- Some resolvers seem to sort round-robin results alphabetically, distorting the usage
- So, we were benefiting by offloading network, but UX was not as good as we would want

Growing Pains

- External providers were cheaper for bw (and sold per TB), spread out geographically and topographically (different providers / transit)
- Manually managed DNS using Bind Views for some geo-intelligence, required many copies of each zone
- Manually marking servers down/up resulted in slow response to state changes, needed to be automated
- Our initial plan for geo-dns and scaling globally was to implement anycast, but we had limited responses, lack of expertise, and some reluctant providers
- Our basic non-dynamic dns weighting technique was to list multiple ips per node in the RR
- This compounded our management problems, and we ran into the dns response size limit (more later)

EDNS0

The original DNS RFC stated that DNS responses larger than 512 bytes should be returned via TCP rather than UDP

In 1999, EDNS0 was introduced, allowing for DNS responses up to 4096 bytes over UDP with fragmentation

gDNSd's $ADDR_LIMIT_V4 and $ADDR_LIMIT_V6 allow us to limit the size of the response to avoid complications

Old Firewalls Haunt the Internet

Some dated firewall rulesets still block UDP DNS response packets larger than 512 bytes. This is not normally a problem because larger responses are only returned to clients who indicate support for EDNS0, however if an intermediary firewall blocks DNS responses larger than 512, the user requests a larger response but then cannot receive it, causing the lookup to time out. DNSSEC will make this issue more pronounced, and possibly result in the issue being rectified

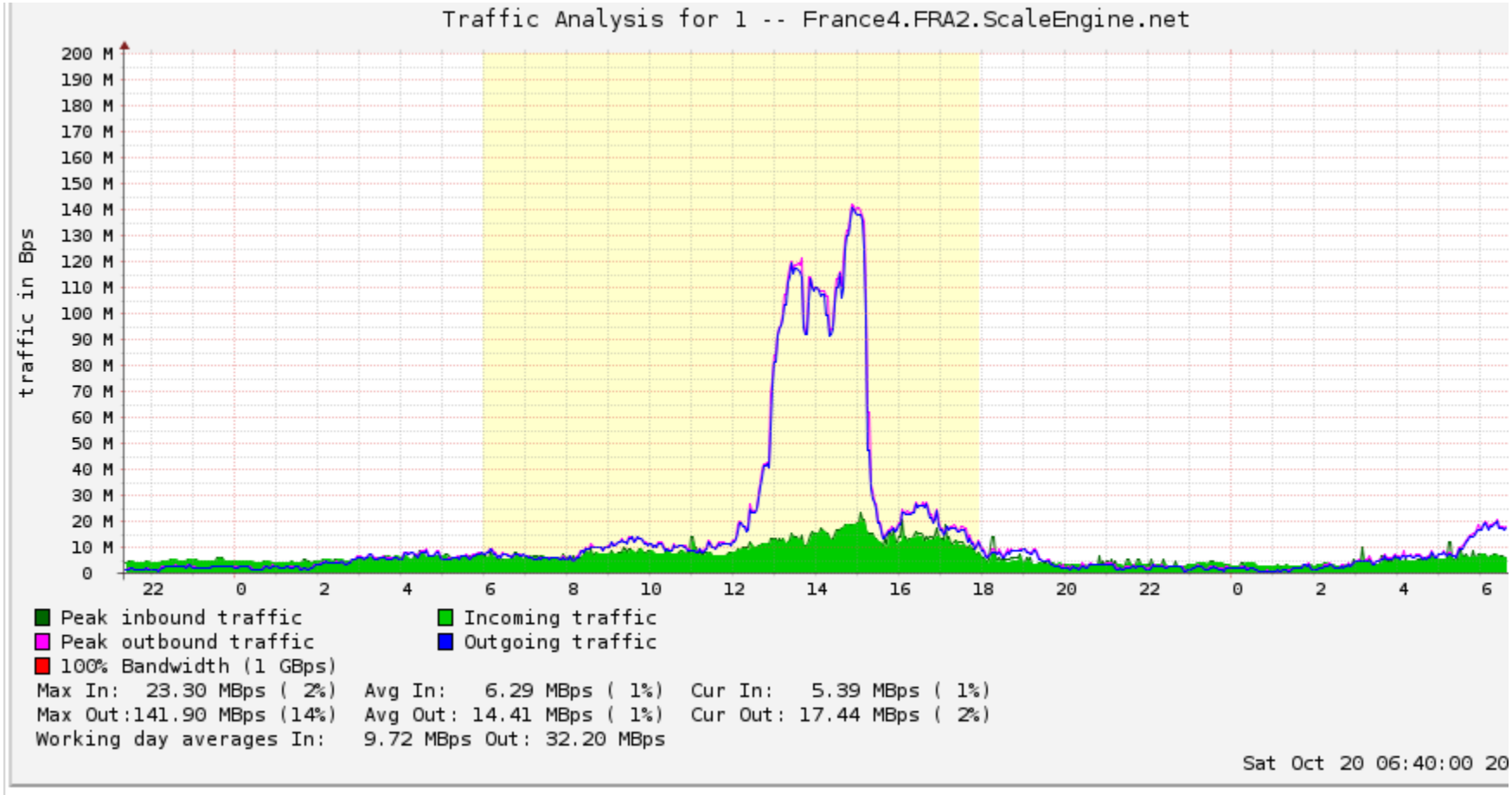
New Territory - Video

- We got requests to do video streaming, and started playing around with it
- More and more requests came from Europe
- Entirely different scaling problems now, with video it was link capacity that become an issue
- Needed gigabit servers, and providers that would sell us additional bandwidth at a reasonable price
- Surprise, Bandwidth is even cheaper in Europe!
- Some providers in Europe had bad transit to NA
- Completely unpredictable scaling demand compared to HTTP
- Video requires dedicated bandwidth, no contention

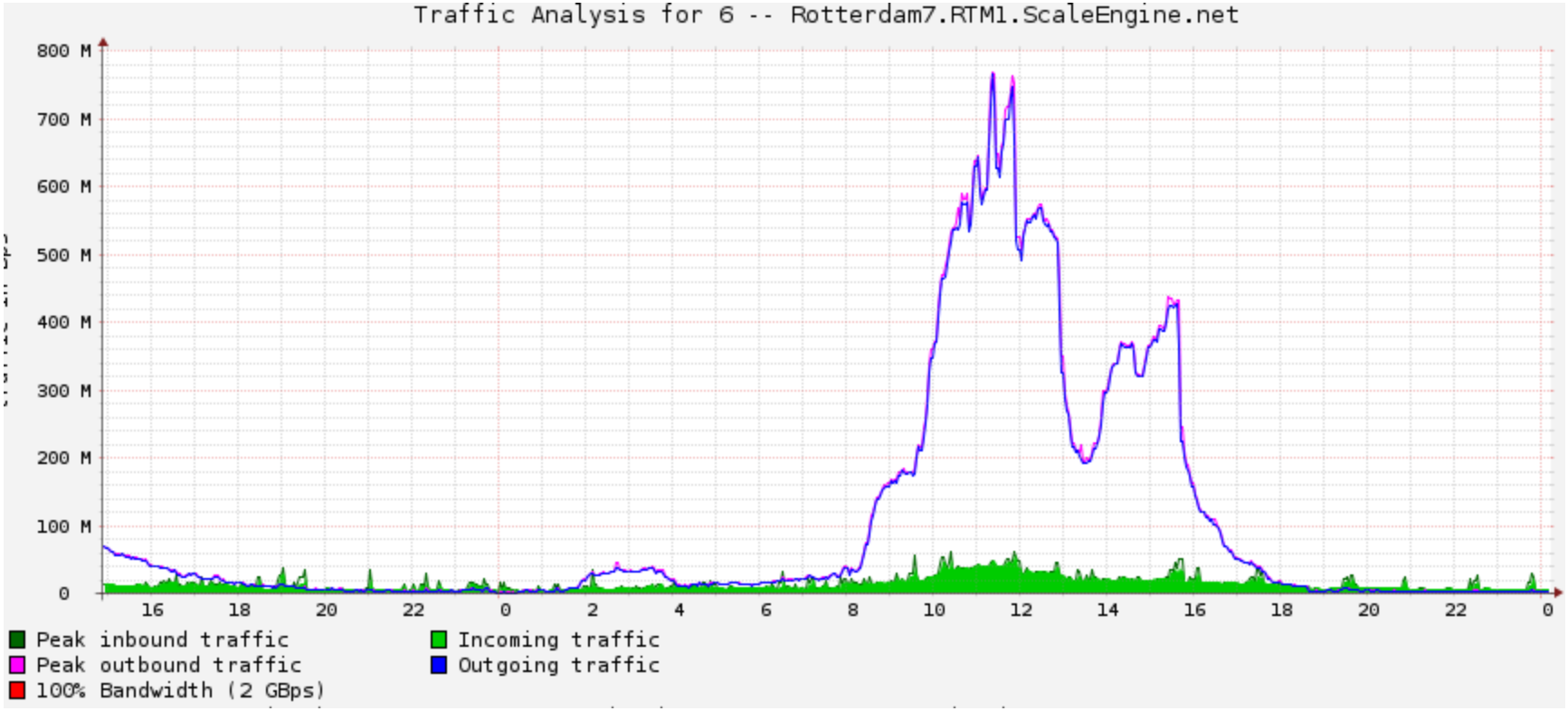
Video Challenges

- News and sports video can spike very quickly (Airport emergency in Iceland)
- 'Fair' load balancing of viewers didn't make sense because different streams had different bitrates, some were audio only, some servers have more capacity
- We wanted to factor geography into the load balancing
- Very important to not send viewers to overloaded servers, breaks stream for existing viewers
- Therefore, measuring network and application health is essential
- Prefer to deliver from closer nodes, but not married to it
- We don't control the customer's app, so layer7 balancing was not really an option
- How do we apply these business rules to DNS?

spiky video graph



very spiky video graph



What is a GSLB?

A Global Server Load Balancer handles the direction of traffic to different nodes with a focus on geo-location, and (depending on the vendor) a "bunch of talk" about high availability and optimal response time.

In our case, the GSLB routes traffic to edge servers, near the requestor, to provide lower latency and spread load between a number of data centers, as well as automatically diverting traffic from downed servers

What Is Out There Now?

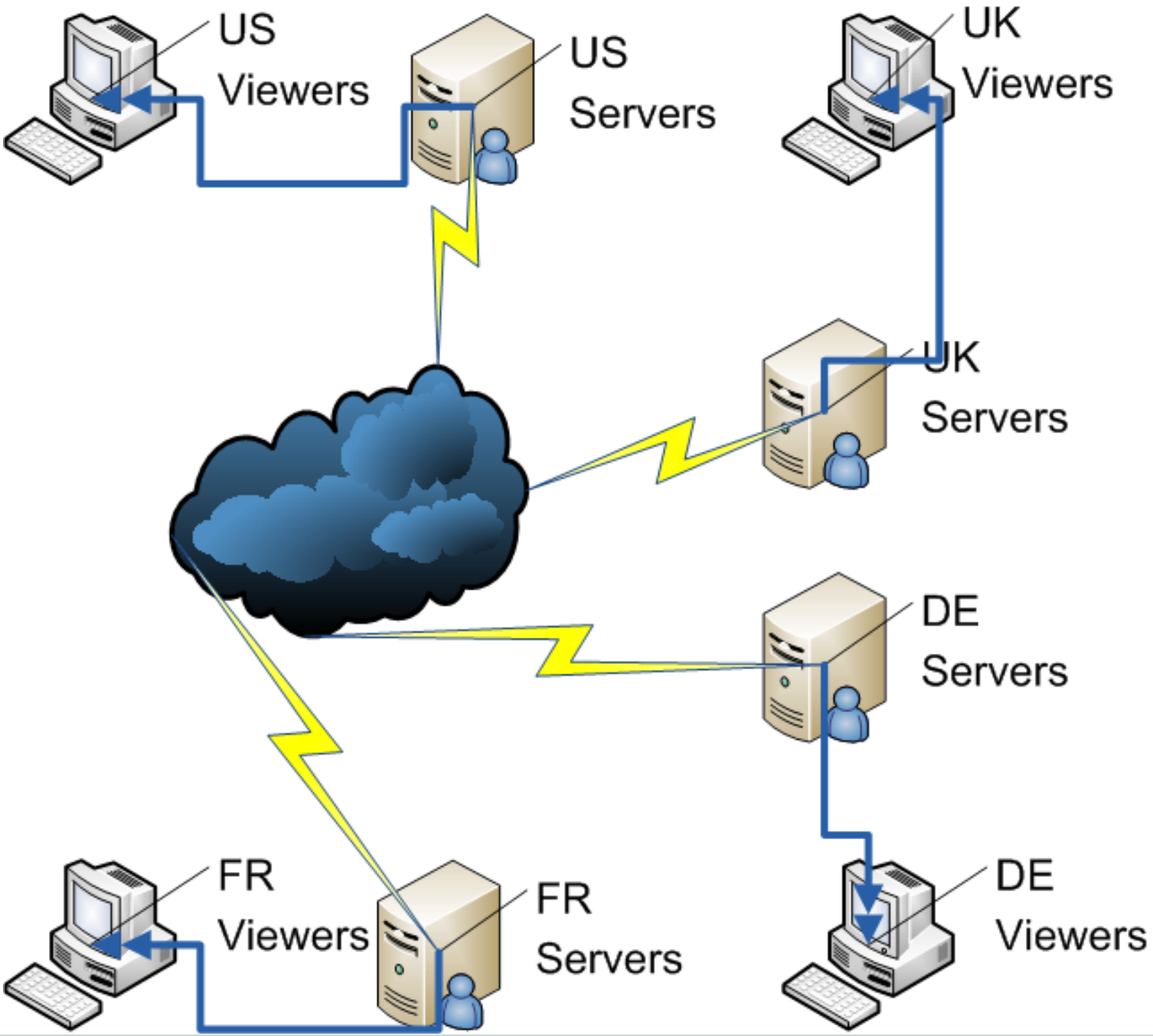
- Commercial Vendor Solutions such as Barracuda Networks - Requires a device and subscription at each location, configuration changes must be manually input on each device by an operator.
- Response Policies are quite limited: either Region, or GeoIP
- Cumbersome non-automated updating
- Sometimes limited monitoring
- Not open, closed version of Linux
- Expensive

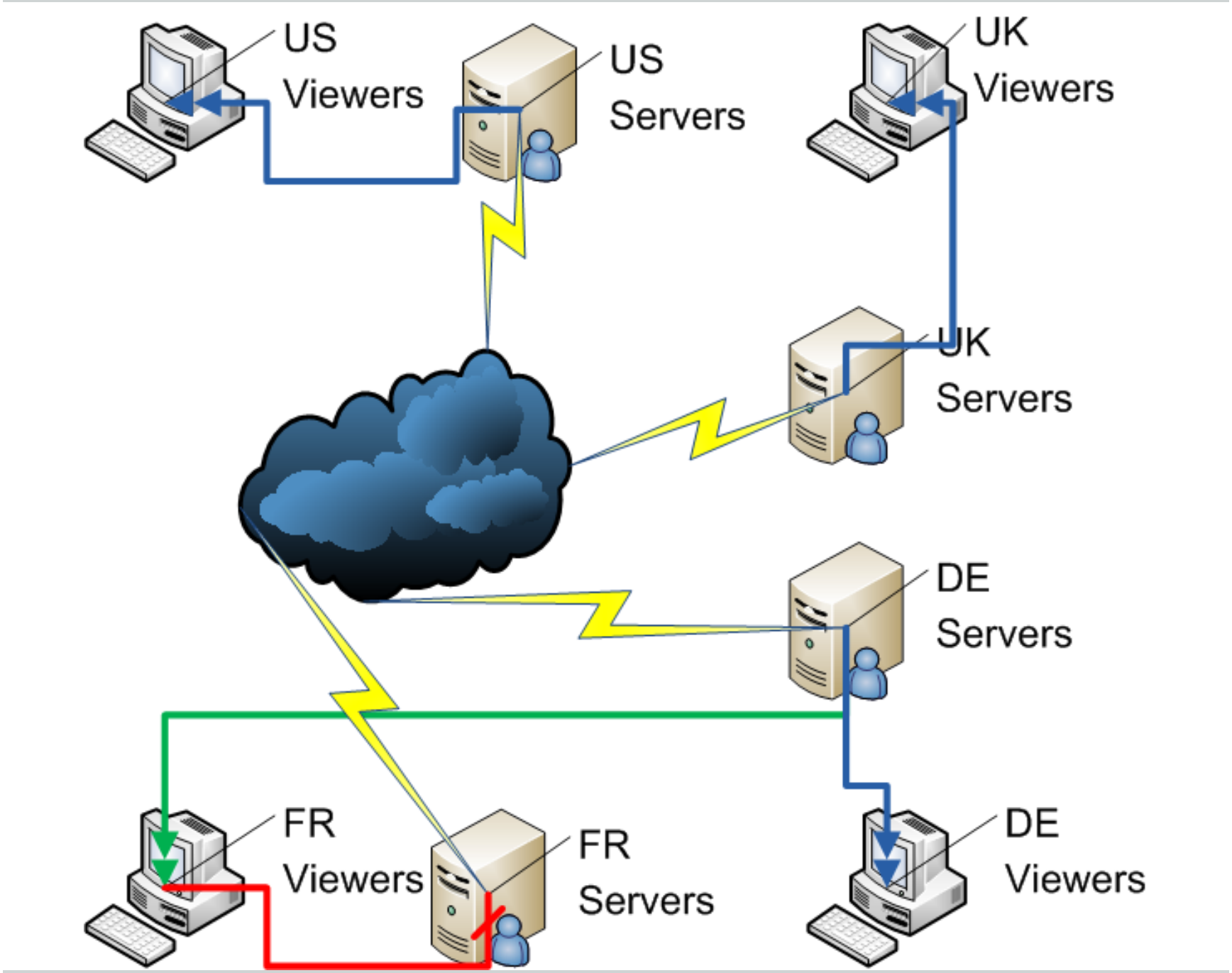
Open Source

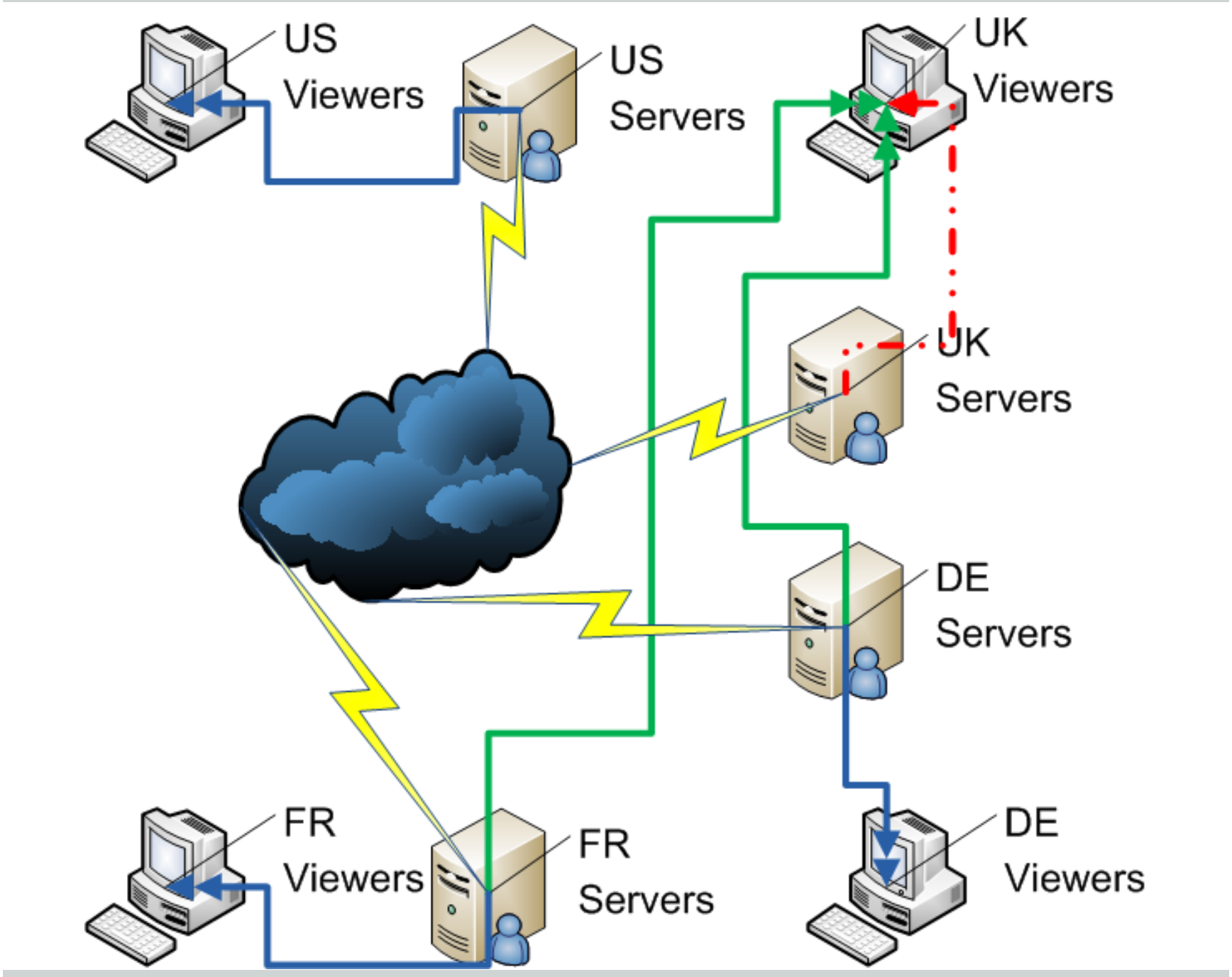
- Bind9 with Views - Summarize subnets from a GeoIP database and create views, uses excessive amounts of memory and takes a while to reload config, requires separate zone file for each view, all zones must be in views, only considers resolver IP, breaks master-slave replication, no monitoring
- Bind9 with GeoIP Patch - Interface with GeoIP via C API, slow to patch for new Bind releases, only considers resolver IP, no monitoring
- PowerDNS w/ Geo Backend - Only reads rbldnsd GeoIP maps
- Anycast - limits flexibility, not application aware without effort, so you dont get automated pool management

The Current Solutions

	Layer	Cost	Manageability	Scalability	Configurability	EDNS-CS
Barracuda	TCP	*	*	***	***	
Bind9 w/ Views	DNS	*****	*	*	*	
Bind9 w/ GeoIP	DNS	*****	**	*	**	
PowerDNS w/ GeoIP	DNS	****	**	*	**	
Anycast	IP	**	**	***	*	
gDNSd	DNS	*****	****	*****	*****	


US
Viewers
US
Servers
UK
Viewers
UK
Servers
DE
Servers
FR
Viewers
FR
Servers
DE
Viewers


US
Viewers
US
Servers
UK
Viewers
UK
Servers
DE
Servers
FR
Viewers
FR
Servers
DE
Viewers


US
Viewers
US
Servers
UK
Viewers
UK
Servers
DE
Servers
FR
Viewers
FR
Servers
DE
Viewers

gDNSd to the rescue

- Fast authoritative only DNS server
- Supports EDNS Client Subnet draft
- Reads Maxmind GeoIP binary databases
- Monitoring plugins, flapping detection
- Response plugins
  - Simple Failover
  - Multi-Failover
  - Weighted Response
  - Geographic Multi-failover
- Extension of standard BIND zone files
- Limit number of addresses in response
- We created dns/gdnsd because it wasn't there

gDNSd zones

- Zones are typical BIND zones (only some record types are supported)
- Two important new types, DYNA and DYNC
- DYNA records take a plugin name and resource name, and returns 1 or more IP addresses
- DYNC records take a plugin name and resource record, and returns a CNAME
- Supports Includes and response record size limiting

Response Policy

A number of factors can go into defining a Response Policy:

- Active/Passive Failover, or Active/Active load balancing?
- Binary Node Health: Up/Down
- Advanced Node Health: CPU, I/O or Net
- Requestor Location
- Which nodes to use when
- How many nodes to return in a response

gDNSd Basic Plugin Config

```
multifo => {
  up_thresh => 0.3,
  service_types => default,
  pubwww => {
    service_types => [ corpwww_type, default ],
    addrs_v4 => [ 192.0.2.100, 192.0.2.101, 192.0.2.102 ]
    addrs_v6 => {
      service_types => [ up ],
      up_thresh => 0.7
      lb01_v6 => 2001:DB8::1,
      lb02_v6 => 2001:DB8::2,
      lb03_v6 => 2001:DB8::3,
    }
  }
  v4www => {
    lb01 => 192.0.2.200,
    lb02 => 192.0.2.201,
    lb03 => 192.0.2.202,
  }
}
```

GeoDNS Gotchas

- GeoIP data is not always correct
- The source IP of DNS requests is not the client, but their resolver, usually their ISP, but maybe also Google or OpenDNS
  - EDNS0-Client-Subnet an IETF draft to have recursive servers include the /24 of the requesting client as part of the DNS request, gDNSd can read this any do GeoIP on the client ip instead of the resolver ip
  - Supported by Google DNS and OpenDNS but requires you be on their whitelist
  - Google's servers use Anycast, requests actually come from a unicast ip, but GeoIP places them all at the GooglePlex in California, even when they are in Europe
  - We use manual overrides to map Google and OpenDNS resolver IPs to their actual location

gDNSd GeoIP Config (Auto)

```
geoip => { maps => { se-automap => {
        geoip_db => /usr/local/var/gdnsd/GeoLiteCity.dat
        datacenters => [ TOR1-1, SEA1-1, LAX1-1, PHX1-1, CHI1-1, CHI2-1, DFW1-1,
        ATL1-1, NYC1-1, FRA1-1, FRA2-1, DEU1-1, AMS1-1, AMS2-1, RTM1-1 ]
        auto_dc_coords => {
        TOR1-1 => [ 43.6532260 , -79.3831843 ], SEA1-1 => [ 47.4938270 , -122.2933670 ],
        LAX1-1 => [ 34.0482680 , -118.2549910 ], PHX1-1 => [ 33.4159838 , -112.0084724 ],
        CHI1-1 => [ 41.9898460 , -87.9564730 ], CHI2-1 => [ 41.8781136 , -87.6297982 ],
        DFW1-1 => [ 32.8395269 , -96.8649300 ], ATL1-1 => [ 33.7555614 , -84.3914132 ],
        NYC1-1 => [ 40.7084175 , -74.0071869 ], FRA1-1 => [ 48.8566140 , 2.3522219 ],
        DEU1-1 => [ 49.4451843 , 11.0874220 ], AMS1-1 => [ 52.3702157 , 4.8951679 ],
        AMS2-1 => [ 52.3702157 , 4.8951679 ], RTM1-1 => [ 51.9242160 , 4.4817760 ],
        }
        nets => {
        65.39.148.0/28 => [ TOR1-1, CHI1-1 ]
        }
} } }
```

gdns GeoIP Config (advanced)

```
geoip => { maps => { se-region-map => {
    geoip_db = /usr/local/var/gdnsd/GeoLiteCity.dat
    datacenters => [ DC-AF, DC-AS, DC-EU, DC-NA, DC-OTHER
        DC-EU-DE, DC-EU-FR, DC-EU-NL, DC-EU-UK, DC-NA-CA ]
    map => {
        AF => { default => [ DC-EU ] }
        AS => { default => [ DC-AS, DC-OTHER ] }
        EU => {
                DE => [ DC-EU-DE, DC-EU ], FR => [ DC-EU-FR, DC-EU ],
                NL => [ DC-EU-NL, DC-EU ], GB => [ DC-EU-UK, DC-EU ],
                default => [ DC-EU ]
        }
        NA => { CA => [ DC-NA-CA, DC-NA ],    default => [ DC-NA ] }
        default => [ DC-OTHER ]
        }
} } }
```

gDNSd GeoIP Config (continued)

```
resources => { se-full-esc => { map => se-region-map,
    service_types => [ varnish ],
    dcmap => {
        DC-NA => {
        varnish_sea1-1_a = 64.120.19.11, varnish_lax1-1_a = 64.120.9.155
        varnish_phx1-1_a = 108.62.112.147, varnish_chi1-1_a = 23.19.122.43
        varnish_dfw1-1_a = 173.208.17.139, varnish_nyc1-1_a = 173.208.58.19
        },
        DC-EU => {
        varnish_fra1-1_a = 94.23.84.74, varnish_deu1-1_a = 188.40.87.24
        varnish_ams1-1_a = 94.100.21.179, varnish_ams2-1_a = 213.152.180.200
        varnish_rtm1-1_a = 213.163.66.251, varnish_rtm1-1_b = 213.163.66.251
        },
    }
} }
```

Monitoring

```
service_types => {
    varnish => {
        vhost = "www.appfail.com"
        url_path = /
        port = 80
        interval = 90
        timeout = 6
        up_thresh = 10          #10 good checks in a row to go from DOWN to UP
        ok_thresh = 5           #5 good checks in a row to go from DANGER to UP
        down_thresh = 2         #2 fails moved server from DANGER to DOWN
        plugin = http_status
    }
}
```

Taking monitoring to the next step

We created custom HTTP responders to point gDNSd monitoring at

- Video Servers return an XML document detailing viewers and bandwidth usage
- If bandwidth is above a configured threshold, HTTP response code is 500 instead of 200
- New viewers are directed to servers that are not over threshold
- gDNSd 'up_thres' causes all servers to be returned if all servers are above threshold (fallback for false positives or huge load)
- CDN node responder checks disk i/o which is high after a restart (cache warming is expensive)

Running out of Bandwidth

This all works great, downed and loaded servers come out of the pool, and users generally get a good experience.

When traffic is very high, gDNSd adds all servers back into the pool, our thresholds are conservative because packet loss kills video, so this works for a while

gDNSd can also fail over to different regions, if we are out of bandwidth in Europe, use the North American servers

Eventually, we are just out of servers, now what?

Amazon Web Services

Amazon EC2 to the rescue

The HATE Algorithm
"High Availability Through EC2"

The Amazon HATE Algorithm

We HATE to use Amazon because they are very expensive, but we hate being down even more

Capacity Manager script polls the stats interface on each gDNSd node. Servers and resources are named according to our geographic naming scheme

- We are out of Capacity in EU, spin up EC2s in EU zone
- Elastic IPs are preconfigured in DC-of-last-resort
- Load shifts to EC2s, anti-flapping keeps our servers from being 'up' again, servers must pass low-traffic threshold to return to 'up' status
- Once our servers are back up, Capacity Manager polls Amazon for running instances, check load/viewer count on each, once this is low, terminate instance

Conclusions

- distance (latency) adds to object delivery delay, multiple round trips, performance for HTTP objects suffers
- Video performance lags if tcp has to retransmit, rtmp clients and iOS HLS clients buffer and delay relative to real event time
- proximity to client helps for HTTP, especially varnish (low app latency)
- proximity is less important with video but still nice to have
- video is more about transit quality to avoid retransmits, buffering and lag
- cache warming is slow, performance is uneven

More Conclusions

- Because our expertise is in running freebsd servers, not border routers, an application layer solution is an appropriate limitation on solution
- With GSLB, having various network transit providers becomes an advantage instead of a limitation
- automatically recover from overloaded server or provider transit issues
- complete global view, pools per service, pool 'stages'
- We are able to automate pool management
- with some effort, application health and server network utilization become pool membership criteria