Jan 12 2018

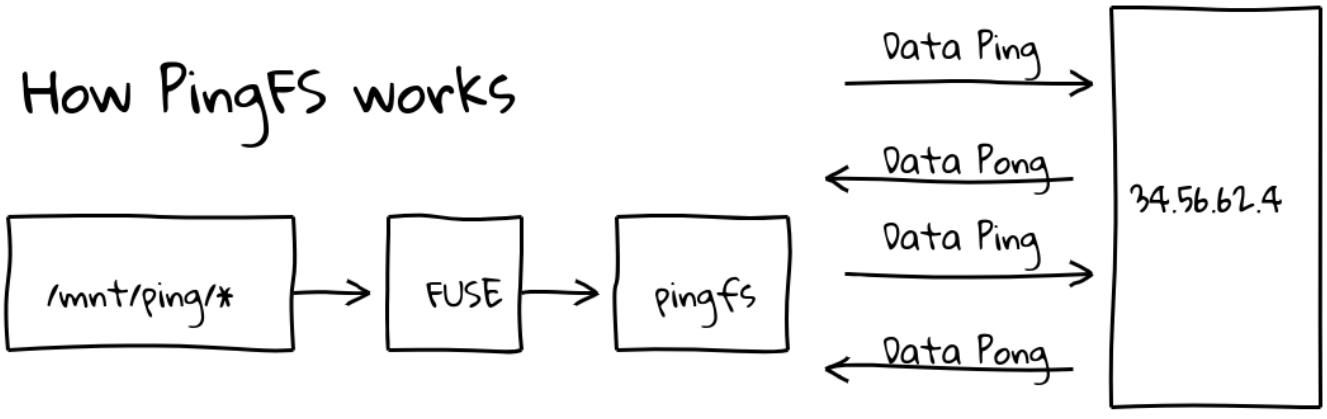
DNSFS. Store your files in others DNS resolver caches

A while ago I did a blog post about how long DNS resolvers hold results in cache for, using RIPE Atlas probes testing against their default resolvers (in a lot of cases, the DNS cache on their modem/router).

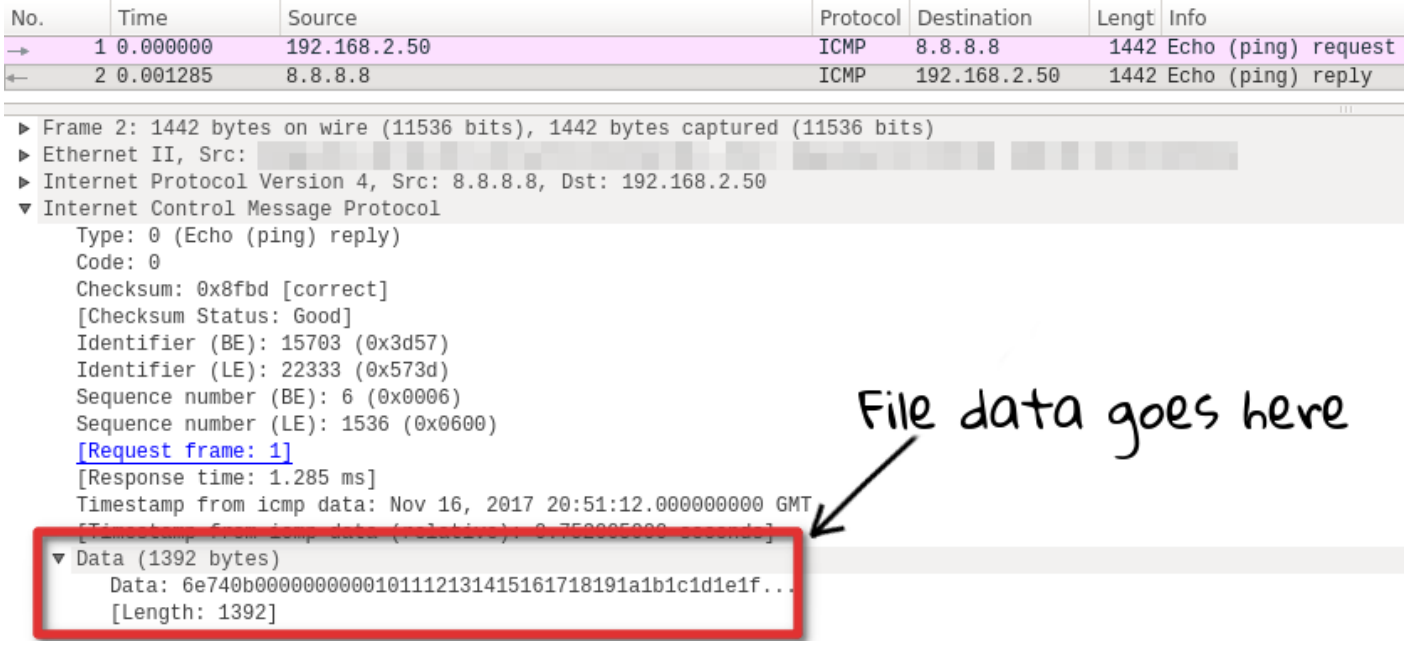
That showed that some resolvers will hold DNS cache entries for a whole week if asked to (https://blog.benjojo.co.uk/post/dns-resolvers-ttl-lasts-over-one-week), and I joked at the end that one could use this for file storage.

Well, I could not stop thinking about doing this. There are surely a lot of open DNS resolvers out on the internet, that are just *asking* to be used for storing random things in them. Think of it. Possibly tens of gigabytes of cache space that could be used!

This is not the first time something like this has been done, Erik Ekman made PingFS, a file system that stores data *in the internet itself* .



This works because inside every ping packet is a section of data that must be sent back to the system that sent the ping, called the data payload:



Because you can put up to 1400-ish bytes in this payload, and pings take time to come back, you can use the speed of light in fiber as actual storage.

Now obviously this is not a great idea for long term data storage, since you have to keep transmitting and receiving the same packets over and over again, plus the internet gives no promise that the packet won't be dropped at any time, and if that happens then the data is lost.

However. DNS has caches. It has caches *everywhere*.

This means that the DNSFS looks a lot of the same as PingFS, but once a query is sent it should be cached in the resolver, meaning you don't have to keep sending packets to keep the data alive!

Resolver strategy

For this to work we need a lot of open DNS resolvers. Technically DNS resolvers (except the official ones that a ISP gives out) should be firewalled off from the public internet because they are a DDoS reflection risk , but a lot of devices out there ship with bad default configuration that allows their built in DNS resolvers to be reachable from outside the LAN.

The more open DNS resolvers there are, the more redundancy (or storage space) we have.

For this, we need to scan the whole internet for resolvers. This is a slightly daunting task, however when you take into account the ranges on the internet that are not routable and ranges of those who do not want to be scanned , it amounts to about 3,969,658,877 IP addresses.

In addition to that we are looking for open resolvers, this means that the DNS server on the other end must be able to look up public domain names, most DNS servers are setup to be authoritative for a single domain name, and can't be used by us for file storage.

Getting a list of DNS resolvers

For this, I am using Robert Graham's masscan to send DNS queries to all applicable IP addresses on the internet.

```
ben@metropolis:~$ ./bin/masscan \
> --interface eth0 \
> --router-mac 0c:c4:7a:8f:6a:77 \
> -pU:53 \<----- Scan on UDP and port 53
> --adapter-ip 185.230.223.69 \
> --exclude-file ./exclude.list \<--- Exclude pointless/disallowed IPs
> --output-format grepable \
> --output-filename DNS-SCAN.list \
> --rotate-size 512m \<---------------- Split the log files into 512MB chunks
> --rate 43 \<-------------- Send at 43 packets per second
> --pcap-payloads single.pcap \
>   0.0.0.0/0                ^-------------- Send a packet from a pcap
    ^-- Scan every IP address
```

However this command has a problem, I am looking for open resolvers, not just things that will reply to port 53 on UDP.

My solution is to use a great feature of the linux kernel called BPF filters (you can read a great article about BPF filters and their use to filter traffic on the Cloudflare blog). You can use them with iptables to drop any traffic you don't want, but programmatically! One BPF rule can do a whole chain worth of work.

I managed to write a tcpdump filter that only matched the DNS responses that I wanted (ones with a single results inside them).

```
tcpdump -ni eth0 port 53 and udp and ip[35] != 0x01 and net 185.230.223.69/32
```

I then compiled it to a raw BPF rule using a small helper program:

```
root@xxxx:~/masscan# ./bpf-gen  RAW 'port 53 and udp and ip[35] != 0x01 and
net 185.230.223.69/32'
25,48 0 0 0,84 0 0 240,21 21 0 96,48 0 0 0,84 0 0 240,21 0 18 64,48 0 0 9,21
16 0 132,21 15 0 6,21 0 14 17,40 0 0 6,69 12 0 8191,177 0 0 0,72 0 0 0,21 2
0 53,72 0 0 2,21 0 7 53,48 0 0 35,21 5 0 1,32 0 0 12,21 2 0 3118915397,32 0
0 16,21 0 1 3118915397,6 0 0 65535,6 0 0 0
```

and then inserted it into IPTables:

```
root@xxxx:~/masscan# iptables -I INPUT -m bpf --bytecode "25,48 0 0 0,84 0 0
240,21 21 0 96,48 0 0 0,84 0 0 240,21 0 18 64,48 0 0 9,21 16 0 132,21 15 0
6,21 0 14 17,40 0 0 6,69 12 0 8191,177 0 0 0,72 0 0 0,21 2 0 53,72 0 0 2,21
0 7 53,48 0 0 35,21 5 0 1,32 0 0 12,21 2 0 3118915397,32 0 0 16,21 0 1
3118915397,6 0 0 65535,6 0 0 0" -j DROP
```

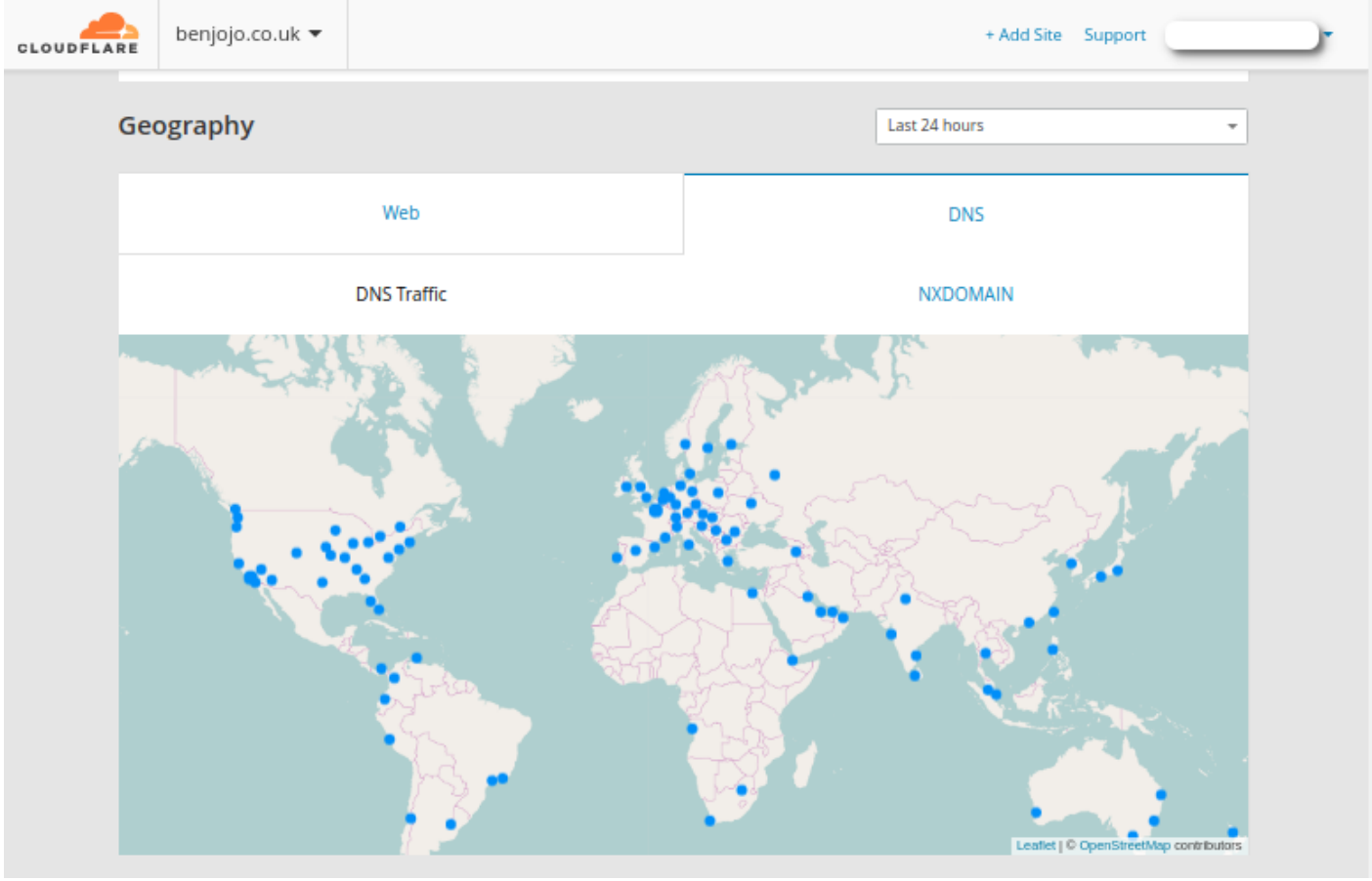
Now masscan will only see and then log results I am interested in. No need to do a 2nd pass to qualify servers.

```
Starting masscan 1.0.4 (http://bit.ly/14GZzcT) at 2017-11-16 22:45:32 GMT
-- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 3969658877 hosts [1 port/host]
rate: 49.35-kpps,  1.11% done,  21:37:45 remaining, found=0
```

After waiting about 24 hours for the scan to complete I got only two abuse notices! One was automated and very formal, the other not so much:

> I think this guy misunderstands that a NOC abuse contact isn't for sending that kind of abuse to pic.twitter.com/t0ZIbgKyai
>
> — Ben Cox (@Benjojo12) November 17, 2017

At the end I was left with **3,878,086** open DNS resolvers, from all ranges and places in the world. Visualised nicely from Cloudflare's DNS traffic analytics:



Basically, where there are Cloudflare data centers, there are open DNS resolvers.

The country breakdown for open resolvers is as follows:

```
ben@metropolis:~/Documents/dnsfs/bulk-mmlookup$ pv ../uniqiplist.txt | bulk-
mmlookup | awk '{$1 = ""; for (i=2; i<NF; i++) printf $i " "; print $NF}' |
sort | uniq -c | sort -n | tac
52.9MiB 0:01:03 [ 850KiB/s] [=======================================>] 100%
1498094 China
 285749 United States
 233549 Republic of Korea
 168979 Russia
 167145 Brazil
 153170 Taiwan
 142655 India
  83581 Italy
  76894 Turkey
  69300 Poland
  62542 Philippines
  53266 Indonesia
  46055 Japan
  43331 Romania
  40434 Bulgaria
  39548 Australia
  36099 Iran
  32996 Canada
  29971 South Africa
```

And ISP:

```
ben@metropolis:~/Documents/dnsfs/bulk-mmlookup$ pv ../uniqiplist.txt | bulk-
mmlookup -type isp | awk '{$1 = ""; for (i=2; i<NF; i++) printf $i " ";
print $NF}' | sort | uniq -c | sort -n  | tac
```

```
 52.9MiB 0:00:15 [3.37MiB/s] [=======================================>] 100%
  830615 4134 No.31,Jin-rong Street
  355713 4837 CHINA UNICOM China169 Backbone
  146755 4766 Korea Telecom
  140772 3462 Data Communication Business Group
   86075 4812 China Telecom (Group)
   67874 9829 National Internet Backbone
   62325 209 Qwest Communications Company, LLC
   59514 9121 Turk Telekom
   56682 3269 Telecom Italia
   55965 9299 Philippine Long Distance Telephone Company
   54555 4847 China Networks Inter-Exchange
   50841 12389 PJSC Rostelecom
   48674 5617 Orange Polska Spolka Akcyjna
   47039 4808 China Unicom Beijing Province Network
   40641 26599 TELEFÔNICA BRASIL S.A
   35171 8866 Vivacom
   33075 5650 Frontier Communications of America, Inc.
   31921 9318 SK Broadband Co Ltd
   30181 8708 RCS & RDS
   29821 9808 Guangdong Mobile Communication Co.Ltd.
   28777 17974 PT Telekomunikasi Indonesia
   28731 7738 Telemar Norte Leste S.A.
   25634 701 MCI Communications Services, Inc. d/b/a Verizon Business
   23598 35819 Bayanat Al-Oula For Network Services
   23427 26615 Tim Celular S.A.
   23322 42610 PJSC Rostelecom
   22488 7018 AT&T Services, Inc.
   17009 4713 NTT Communications Corporation
   14707 12880 Information Technology Company (ITC)
   14259 24560 Bharti Airtel Ltd., Telemedia Services
   13989 14420 CORPORACION NACIONAL DE TELECOMUNICACIONES - CNT EP
   13462 7303 Telecom Argentina S.A.
   12069 6713 Itissalat Al-MAGHRIB
   11865 13999 Mega Cable, S.A. de C.V.
   11439 45899 VNPT Corp
   10711 7922 Comcast Cable Communications, LLC
   10256 28006 CORPORACION NACIONAL DE TELECOMUNICACIONES - CNT EP
    9993 45595 Pakistan Telecom Company Limited
    8896 4739 Internode Pty Ltd
```

However, this data is kind of meaningless. Because all it is really showing is the biggest ISP and countries. So let's try looking at the open resolvers per internet user in a country:

Country	Open Resolvers	Internet users	People per resolver
Saint Kitts and Nevis	729	37210	51.0
Grenada	771	41675	54.1
Marshall Islands	158	10709	67.8
Bulgaria	40434	4155050	102.8
Belize	1063	165014	155.2
Republic of Korea	233549	43274132	185.3
Bermuda	308	60047	195.0
Maldives	968	198071	204.6
Guam	601	124717	207.5
Antigua and Barbuda	262	60306	230.2
Ecuador	28923	7055575	243.9
Romania	43331	11236186	259.3
Hong Kong	18698	5442101	291.1
Barbados	739	228717	309.5
Guyana	954	305007	319.7

Country	Open Resolvers	Internet users	People per resolver
Cayman Islands	136	45038	331.2
Seychelles	158	56168	355.5
Liechtenstein	100	36183	361.8
Tunisia	13733	5472618	398.5
Poland	69300	27922152	402.9
Georgia	5078	2104906	414.5
Malta	787	334056	424.5
Andorra	155	66728	430.5
Moldova	4396	1946111	442.7
Italy	83581	39211518	469.1
China	1498094	721434547	481.6
Gabon	358	182309	509.2

Or, if we strip the countries with less than 1 million internet users:

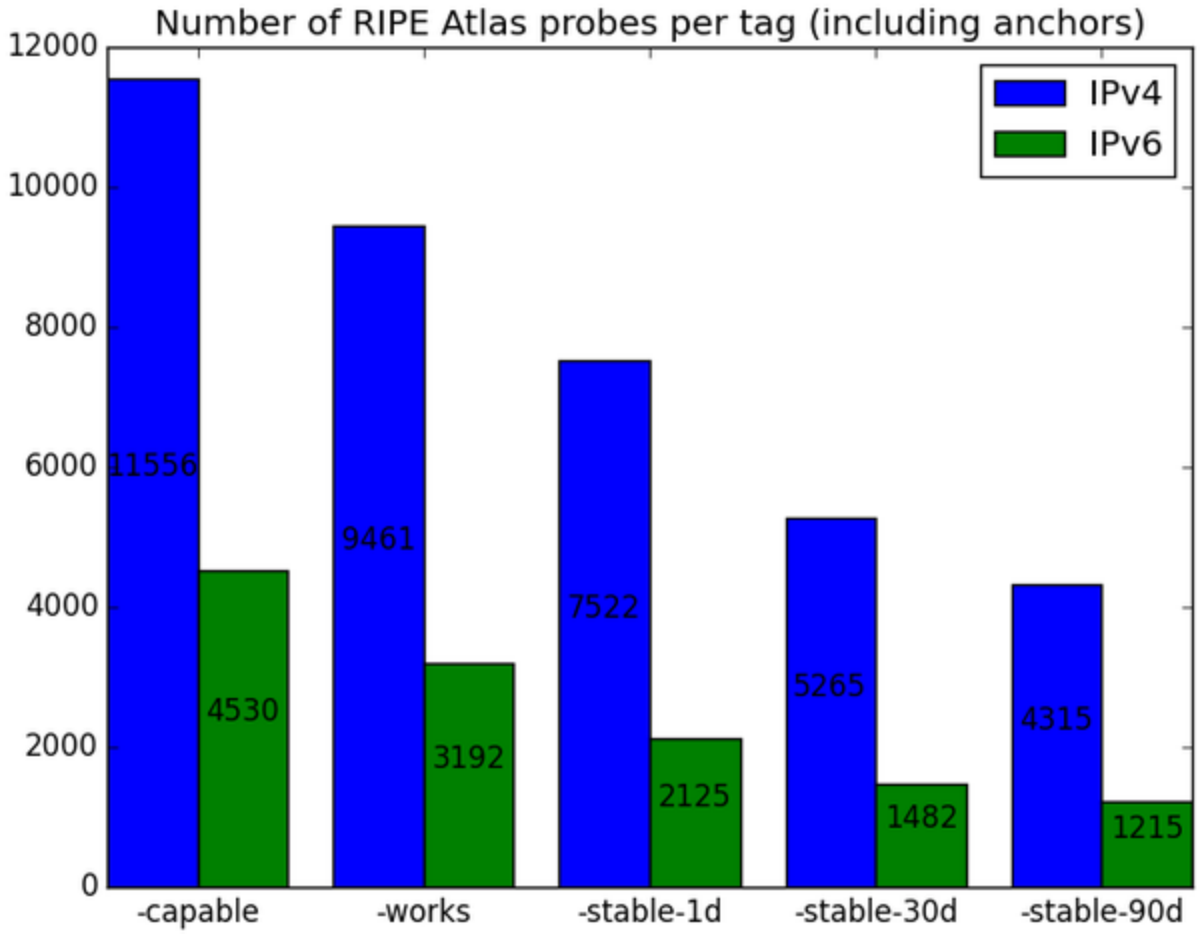
Country	Open Resolvers	Internet users	People per resolver
Bulgaria	40434	4155050	102.8
Republic of Korea	233549	43274132	185.3
Ecuador	28923	7055575	243.9
Romania	43331	11236186	259.3
Hong Kong	18698	5442101	291.1
Tunisia	13733	5472618	398.5
Poland	69300	27922152	402.9
Georgia	5078	2104906	414.5
Moldova	4396	1946111	442.7
Italy	83581	39211518	469.1
China	1498094	721434547	481.6
Australia	39548	20679490	522.9
Turkey	76894	46196720	600.8
Russia	168979	102258256	605.2
Singapore	7552	4699204	622.2
Bolivia	7184	4478400	623.4
Panama	2673	1803261	674.6
Ukraine	27718	19678089	709.9
Philippines	62542	44478808	711.2
Kuwait	4493	3202110	712.7
Lebanon	6333	4545007	717.7
Costa Rica	3486	2738500	785.6
Saudi Arabia	25837	20813695	805.6
Brazil	167145	139111185	832.3
Sweden	10872	9169705	843.4
Uruguay	2654	2238991	843.6
Dominican Republic	6476	5513852	851.4
Morocco	23189	20068556	865.4
Armenia	1743	1510906	866.8

You can find the data yourself here or as a CSV here

How long do resolvers hold cache for?

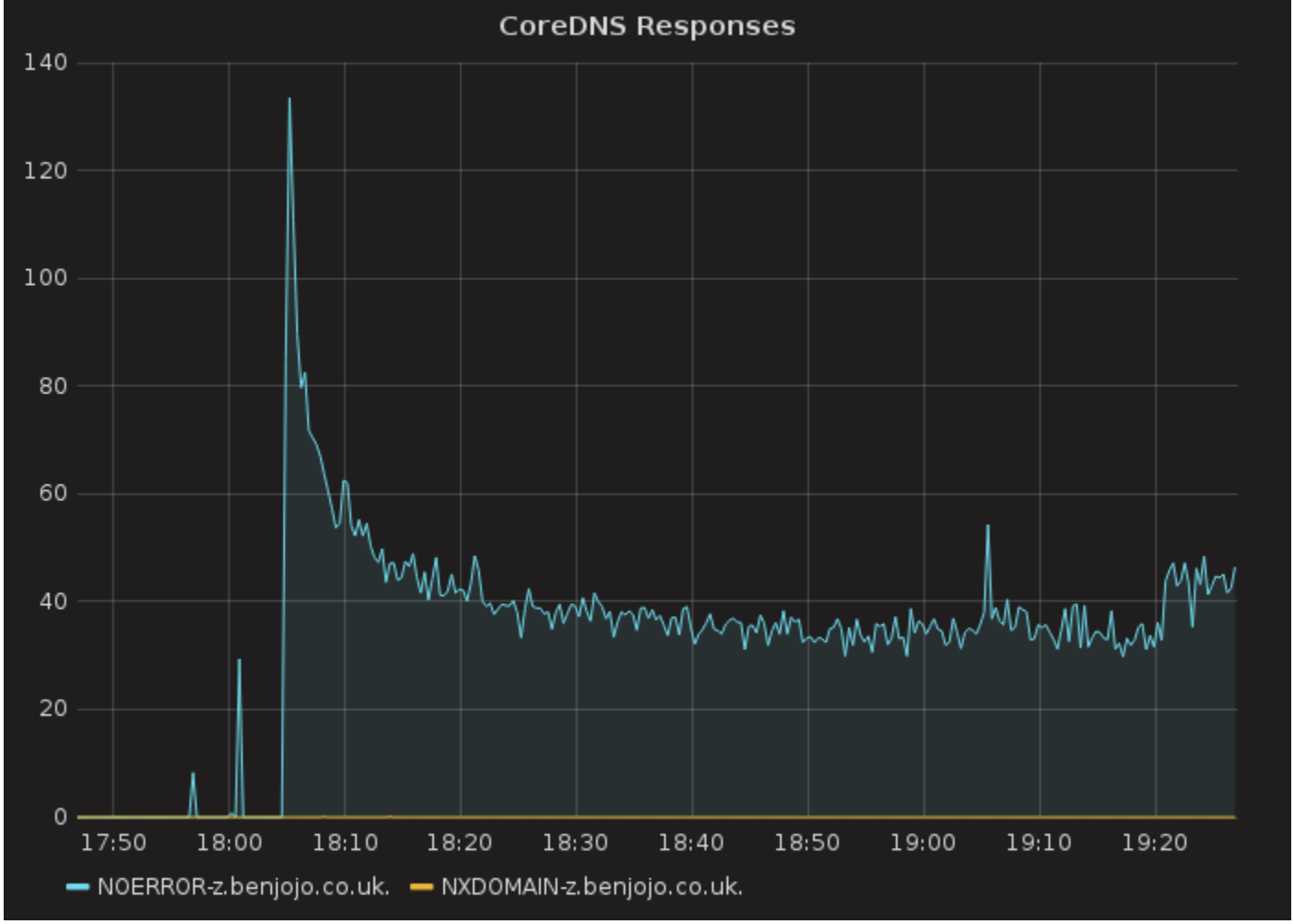
Before testing this, I waited 10 days to allow for all of the resolvers who could be on dynamic IP addresses to change and become unusable. This waiting lost me 37.9% of my IP list.

RIPE Atlas did a decent amount of work to show that a non dismissable percentage of Atlas probes (normally on residential connections) change the public IP address once a day:



After that, I replicated the old method, a very long TTL (in this case, 68 years) and a TXT record containing the current unix timestamp of the DNS server.

The initial query rate on my DNS server was interesting, showing a large initial “rush”, likely due to some DNS resolvers having multiple levels of caches (and thus those upper caches warming up)



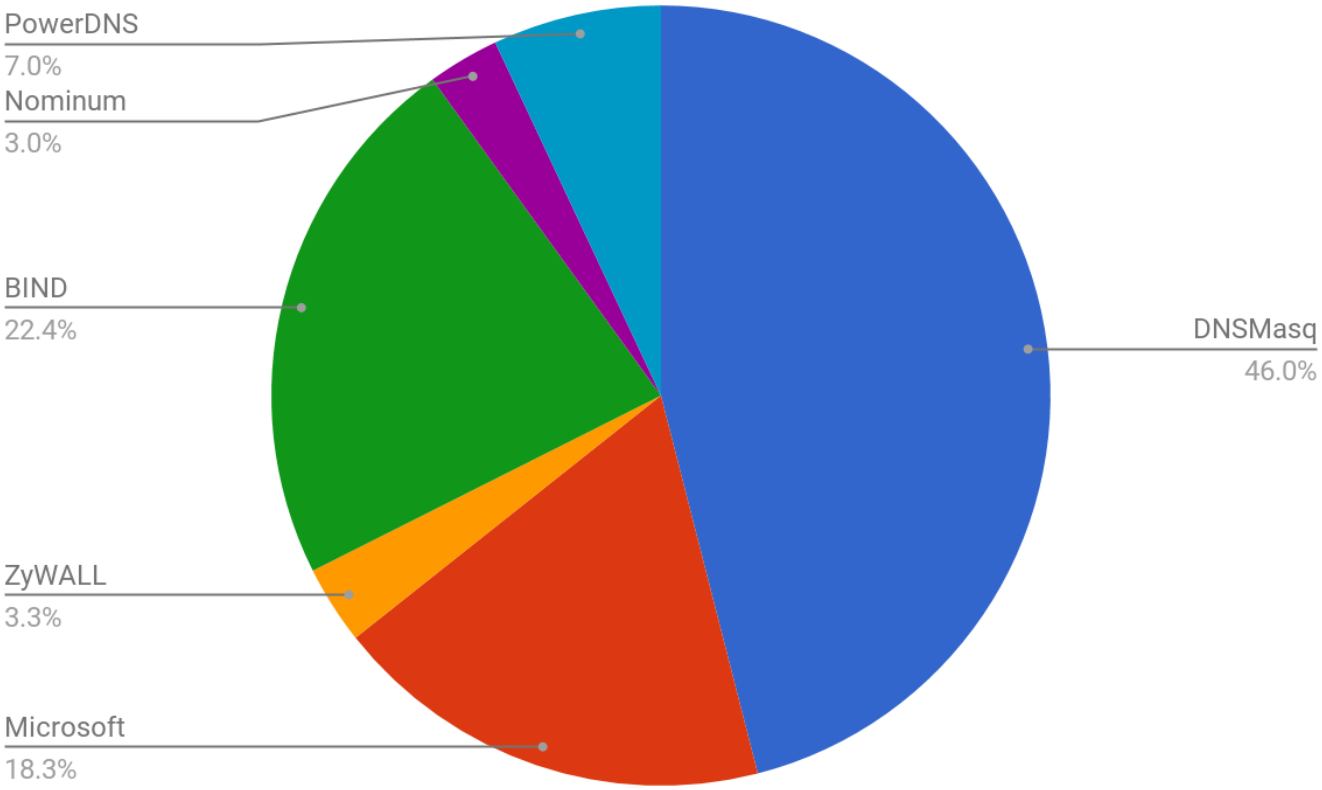
I then queried every server in the list every hour for about a few days using a simple tool link

Then refactored some code on the RIPE atlas post to work with my dataset. This showed that 18% of resolvers can hold items in cache for about a day:

```
ben@metropolis:~/Documents/dnsfs$ cat retention.csv | awk -F ',' '{print
$1}' | grep -v time | ./mm -p -b 6
Values min:0.00 avg:470.30 med=56.00 max:1778.00 dev:635.18 count:2387821
Values:
 value |-------------------------------------------------- percent
     0 |************************************************** 45.28%
     1 |                                                     0.29%
     6 |                                                **   2.08%
    36 |                                           ********** 9.70%
   216 |                           ************************** 24.27%
  1296 |                                 ******************** 18.38%

ben@metropolis:~/Documents/dnsfs$ cat retention.csv | awk -F ',' '{print
$1}' | grep -v time | ./mm -b 6
Values min:0.00 avg:470.30 med=56.00 max:1778.00 dev:635.18 count:2387821
Values:
 value |-------------------------------------------------- count
     0 |************************************************** 1081109
     1 |                                                    6816
     6 |                                                ** 49721
    36 |                                           ********** 231737
   216 |                           ************************** 579613
  1296 |                                 ******************** 438825
```

The next task is to make a basic guess on how big the cache is in these 400k open resolvers. To assist in this guesswork, I queried `version.bind` on every one of these resolvers, filtered down to the major brands of DNS server, and ended up with the following:



The default cache configurations for these devices are as follows:

Brand	Cache Size
DNSMasq	150 Entries
BIND	10MB worth
PowerDNS	2MB worth
Microsoft	Unlimited???
ZyWall	Unknown
Nominum	Unknown

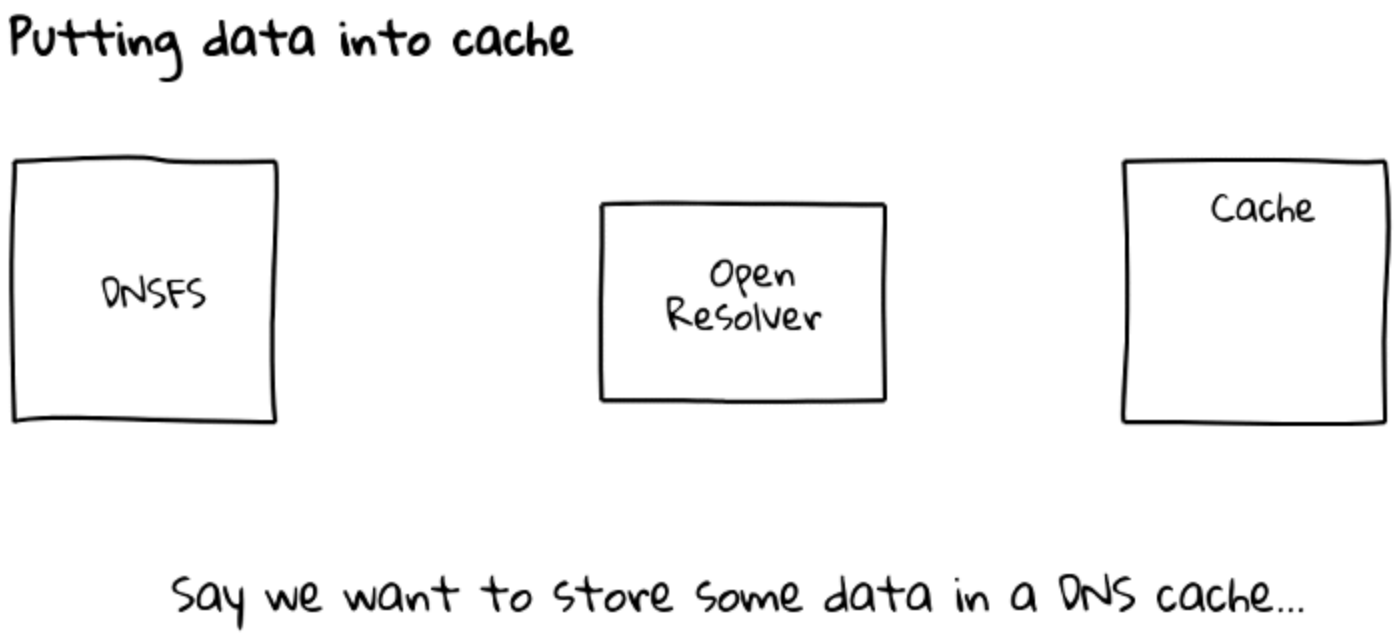
At this point I figured a maximum of 9 TXT records, each 250 character base64 string long (187 bytes ish) would be reasonable.

This means we get approximately (3 * 438825 * 187) = 250~ MB. Assuming we replicate to 3 different resolvers for each TXT record this should also prove “stable” enough to store files for at least a day.

Building the file system

As much as I like FUSE I found that due to modern assumptions of desktop linux it's impractical to use it for high latency backends, which DNSFS would be since its data rate is very low (but still faster than a floppy disk!). So I chose a simple HTTP interface to upload and fetch files to and from the internet.

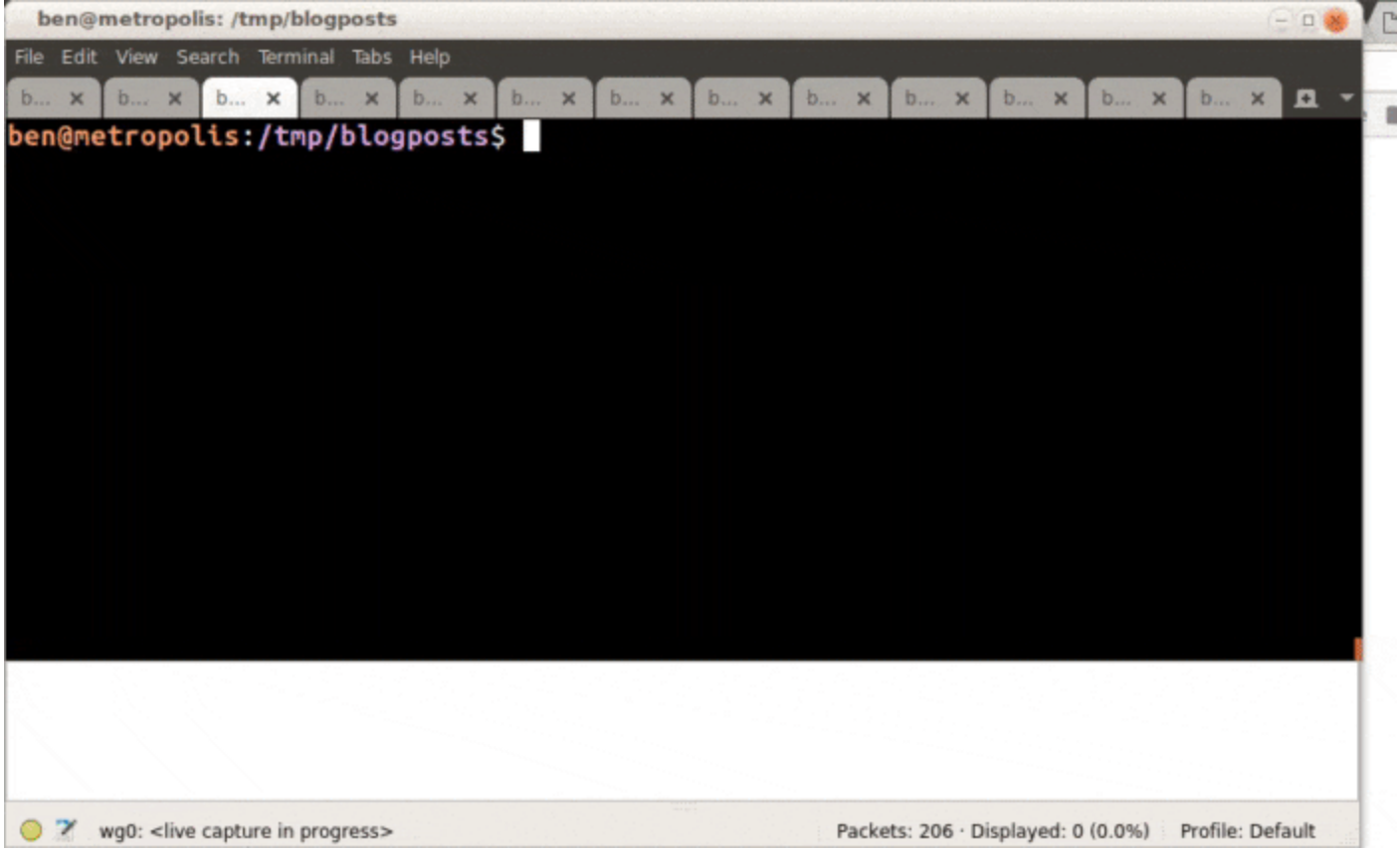
The DNSFS code is a relatively simple system, every file uploaded is split into 180 byte chunks, and those chunks are "set" inside caches by querying the DNSFS node via the public resolver for a TXT record. After a few seconds the data is removed from DNSFS memory and the data is no longer on the client computer.



To spread the storage load, the request filename and chunk number is hashed, then modulated over the resolver list to ensure fair (ish) spread.

Demo

For the demo, I will store one of my previous blog posts in the internet itself!



As you can see, my file has made quite a name for itself.

As always, you can find the code to my madness on my github here: https://github.com/benjojo/dnsfs

And you can follow me on Twitter here for micro doses of madness like this: https://twitter.com/benjojo12

Related Posts:

Building a legacy search engine for a legacy protocol (2017)

TOTP SSH port fluxing (2016)

Random Post:

Teaching a cheap ethernet switch new tricks (2019)