# Drilling the Filter Bubble

## Data-driven optimizations

Rémi Berson

November 14, 2018

Cliqz

# Whoami

Rémi Berson

Cliqz

#Browser, #Search, #Privacy, #Antitracking, #Adblocking

# Plan?

- Cliqz's Adblocker: https://github.com/cliqz-oss/adblocker

# Plan?

- Cliqz's Adblocker: https://github.com/cliqz-oss/adblocker
- Mobiles are *slow*

# Plan?

- Cliqz's Adblocker: https://github.com/cliqz-oss/adblocker
- Mobiles are *slow*
- What can we do?

# Plan?

- Cliqz's Adblocker: https://github.com/cliqz-oss/adblocker
- Mobiles are *slow*
- What can we do?
- Less filters for mobile, based on data (~4k filters)

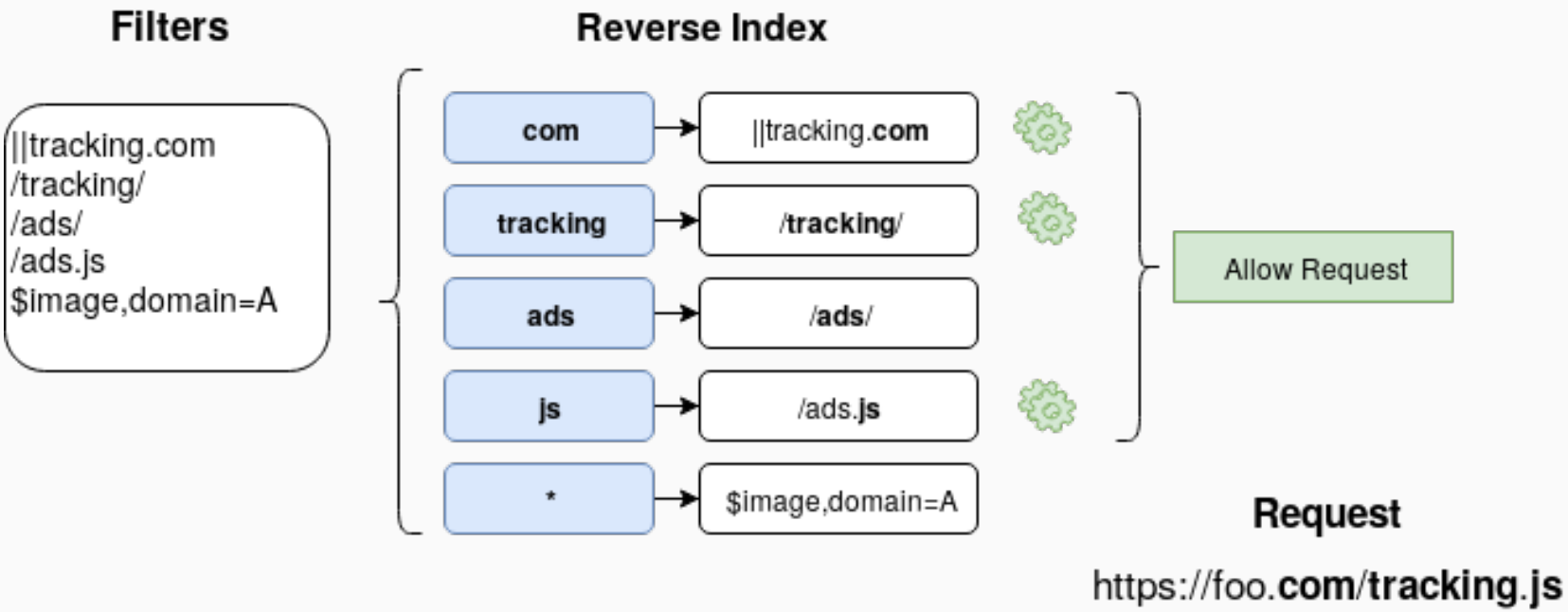
# Data

- 50k top popular domains
- ~25M requests:


```
{
  "url":"https://www.wikipedia.org/.../img/sprite.svg",
  "sourceUrl":"https://www.wikipedia.org/",
  "cpt":"image"
},
{
  "url":"https://www.google.com/images/nav_logo229.png",
  "sourceUrl":"https://www.google.com/?gws_rd=ssl",
  "cpt":"image"
}
```


# Reverse Index - method

# Reverse Index



# Instrumentation

*Measure, then optimize*

For *each bucket*:

1. `token` (e.g.: `https`)
2. number of filters
3. number of “hits”
4. number of “matches”
5. cumulative time spent

# Buckets

No available token (41 filters)

```
$media,domain=damimage.com|imagedecode.com|imageteam.org
$image,third-party,xmlhttprequest,domain=rd.com
$script,domain=zdnet.fr,~first-party
$script,domain=imageporter.com
$script,domain=pornhd.com
$script,domain=imx.to
...
```

# Buckets - http

*Similar filters appear in same buckets*

**http** (161 filters)

```
|http://$image,script,third-party,domain=intoupload.net
|http://$image,script,third-party,domain=linkshrink.net
|http://$image,script,third-party,domain=povw1deo.com
|http://$image,script,third-party,domain=sendit.cloud
|http://$image,script,third-party,domain=dwindly.io
|http://$image,script,third-party,domain=movpod.in
...
```

(https is similar)

# Buckets - ads

*Similar filters appear in same buckets*

**ads** (56 filters)

```
-ads.js?
.com/js/ad.
.com/js/ads/
.com/js/adsense
/ads.js.
/ads.js/*
/ads.js?
/ads/js.
...
```

# Static Optimizations

*While building the index*

# Static Optimizations - Parsing

*Transform some filters while parsing*

- **Avoid RegExps**:
  - `foo*` → `foo`
  - `*bar` → `bar`
  - `||tracking.com^` → `||tracking.com`
- **Avoid string matching**:
  - `|https://` → `$https` (custom option)
  - `|http*://` → `$https,http` (custom options)

# Static Optimizations - Index Domains

*Index filters with no tokens using domains*

This would go in the “catch-all” bucket:

`$image,domain=A|B|C`

Instead, we will index the filter several times in `A`, `B` and `C`.

# Dynamic Optimizations

- On the fly, on “hot” buckets

```
function optimize(filters: Filter[]): Filter[] {
  ...
}
```

# Dynamic Optimizations 1

*Fuse filters with same options and no hostname*

From:

```
-ads.js?
.com/js/ad.
=js_ads&
_ads/js/
_js/ads.js
_js_ads.
_js_ads/
...
```

To one `RegExp`:

```
(-ads.js?)|(.com/js/ad.)|(=js_ads&)|(_ads/js/)|...
```

# Dynamic Optimizations 2

*Fuse filters with same pattern and options but different domains*

From:

```
|https://$script,domain=A|B
|https://$script,domain=B|C
|https://$script,domain=C|D|E|F
```

To only one filter:

```
|https://$script,domain=A|B|C|D|E|F
```

# Dynamic Optimizations 3

*Remove redundant filters*

From:

```
||tracker.com^
||tracker.com/ads
||tracker.com$image
```

To only:

```
||tracker.com^
```

# Results: Requests Filters

*Number of filters checked*

- **blocking**: **10 filters** on average (**18x** improvement)
  - but only **3 filters** required pattern check
- **allowing**: **12 filters** on average (**21x** improvement)
  - but only **7.7 filters** required pattern check

# Results: Requests Timings

*Time to process a request*

- **blocking**: **0.012 ms** on average (**3x** improvement)
  - max: ~1ms
- **allowing**: **0.011 ms** on average (**5x** improvement)
  - max: 300ms => ~2ms (*x150* improvement)

CPU: i7 U-6600 (Skylake)

# Results: Nanopi Neo

*Time to process a request*

- **blocking**: **0.17 ms** on average (14x slower)
  - max: 14ms => ~7ms
- **allowing**: **0.14 ms** on average (12x slower)
  - max: 1500ms => ~7ms

CPU: ARM Cortex-A7 1.2Ghz

# Whotracks.me

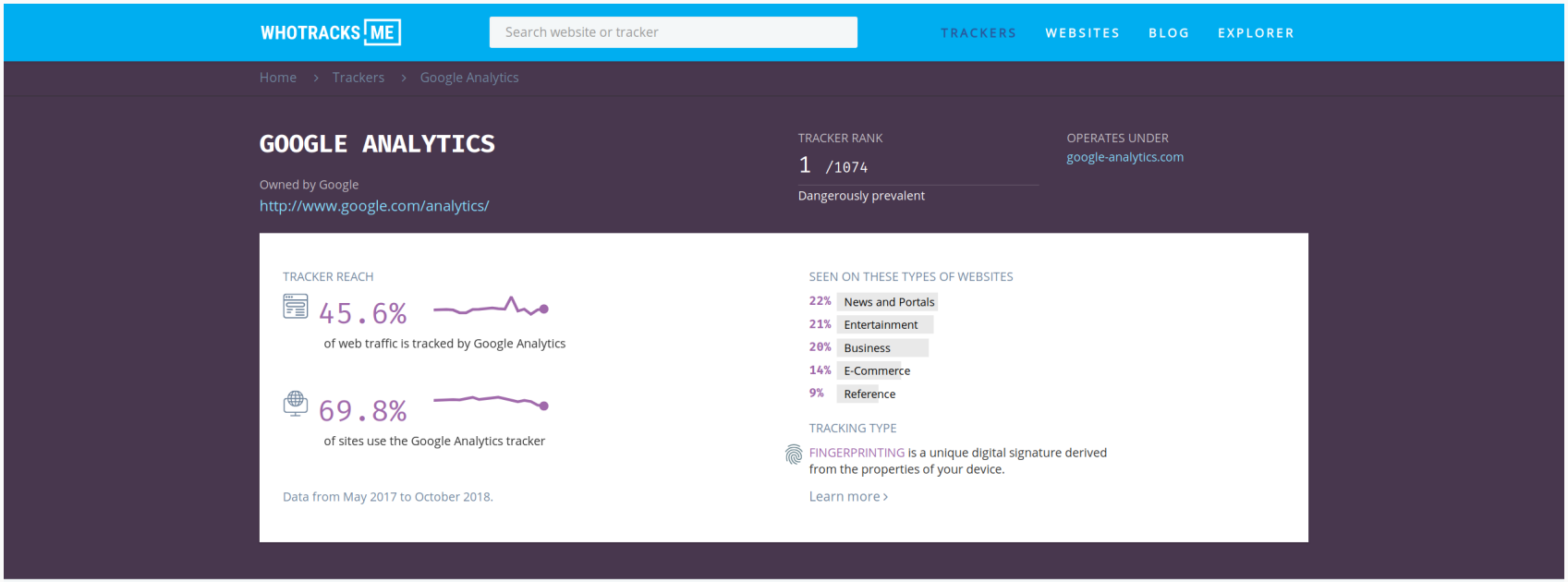



**Figure 1:** Whotracks.me Tracker

# Potential Use-Cases

- Measure the impact of adding a new filter

# Potential Use-Cases

- Measure the impact of adding a new filter
  - Does it block requests on other domains?

# Potential Use-Cases

- Measure the impact of adding a new filter
  - Does it block requests on other domains?
- Optimize coverage of each filter and remove redundancy

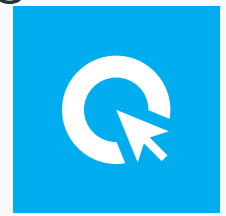
# Potential Use-Cases

- Measure the impact of adding a new filter
  - Does it block requests on other domains?
- Optimize coverage of each filter and remove redundancy
  - Are several filters blocking the same requests?

# Potential Use-Cases

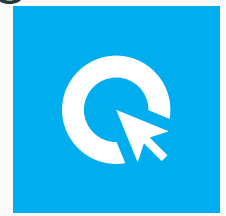
- Measure the impact of adding a new filter
  - Does it block requests on other domains?
- Optimize coverage of each filter and remove redundancy
  - Are several filters blocking the same requests?
- Prioritize loading of filters based on usefulness or create smaller list

- Whotracks.me detect new trackers very fast
- Whotracks.me remove filters targeting non-existant tracking domains



# Potential Use-Cases

- Measure the impact of adding a new filter
  - Does it block requests on other domains?
- Optimize coverage of each filter and remove redundancy
  - Are several filters blocking the same requests?
- Prioritize loading of filters based on usefulness or create smaller list
  - If you could keep only N filters, which one should you pick?
- Whotracks.me detect new trackers very fast
- Whotracks.me remove filters targeting non-existant tracking domains



# Results: Index

*31768 network filters from Easylist*

- 28709 buckets
- 26611 of buckets have only one filter inside (92%)
- 1490 of buckets have two filters (5%)
- 11485 filters were inspected at least once (36%)
- 4336 filters matched a request at least once (13%)