Exploring container image distribution with casync

Experiments with the Content-Addressable Data Synchronization Tool

WE BUILD LINUX TECHNOLOGIES

Hi, I'm Alban

Alban Crequy
CTO @ Kinvolk

alban@kinvolk.io

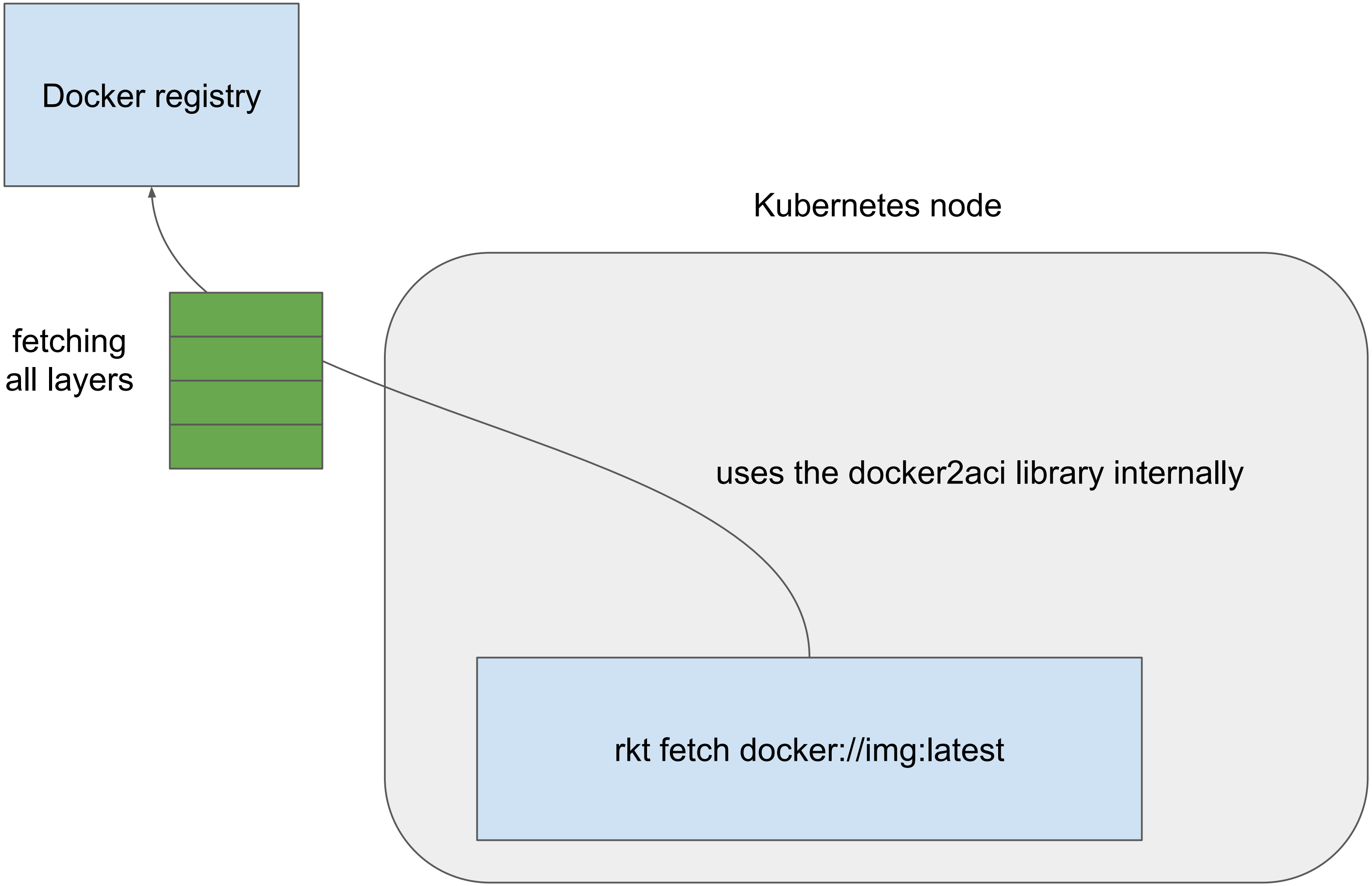
Plan

- ★ Existing container image distribution mechanisms
- ★ Problem statement: wasting network bandwidth
- ★ Exploring two solutions: bittorrent and casync

Container image distribution with rkt

- ★ Support for the Docker registry (for Docker images)
  - ○ https://github.com/docker/distribution
- ★ Support for ACI Discovery (for ACI images)
  - ○ https://github.com/appc/spec/blob/master/spec/discovery.md

Docker registry



Docker registry

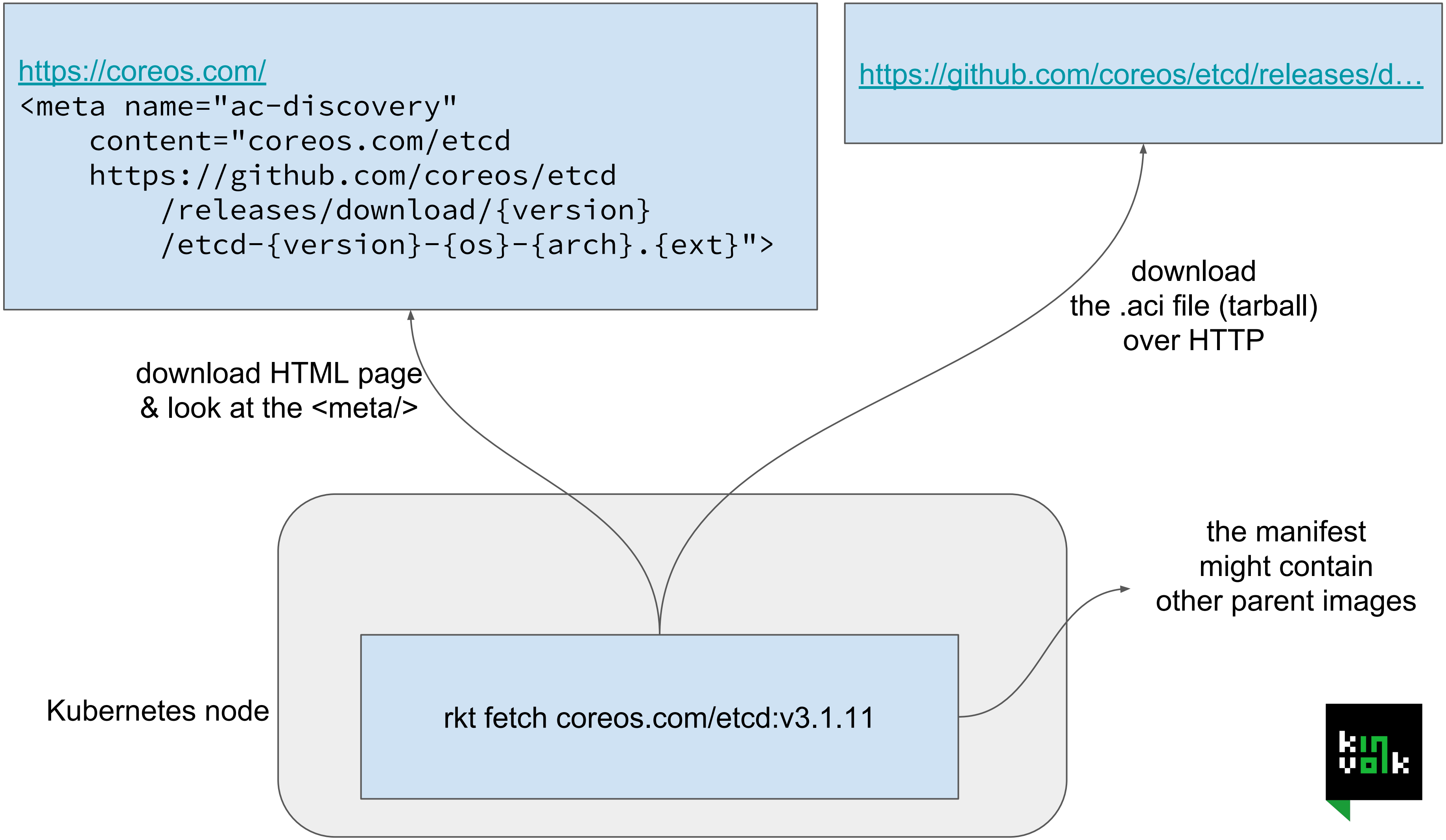
Kubernetes node

fetching all layers

uses the docker2aci library internally

rkt fetch docker://img:latest

ACI Discovery



https://coreos.com/

```
<meta name="ac-discovery"
    content="coreos.com/etcd
    https://github.com/coreos/etcd
        /releases/download/{version}
        /etcd-{version}-{os}-{arch}.{ext}">
```

https://github.com/coreos/etcd/releases/d…

download HTML page
& look at the <meta/>

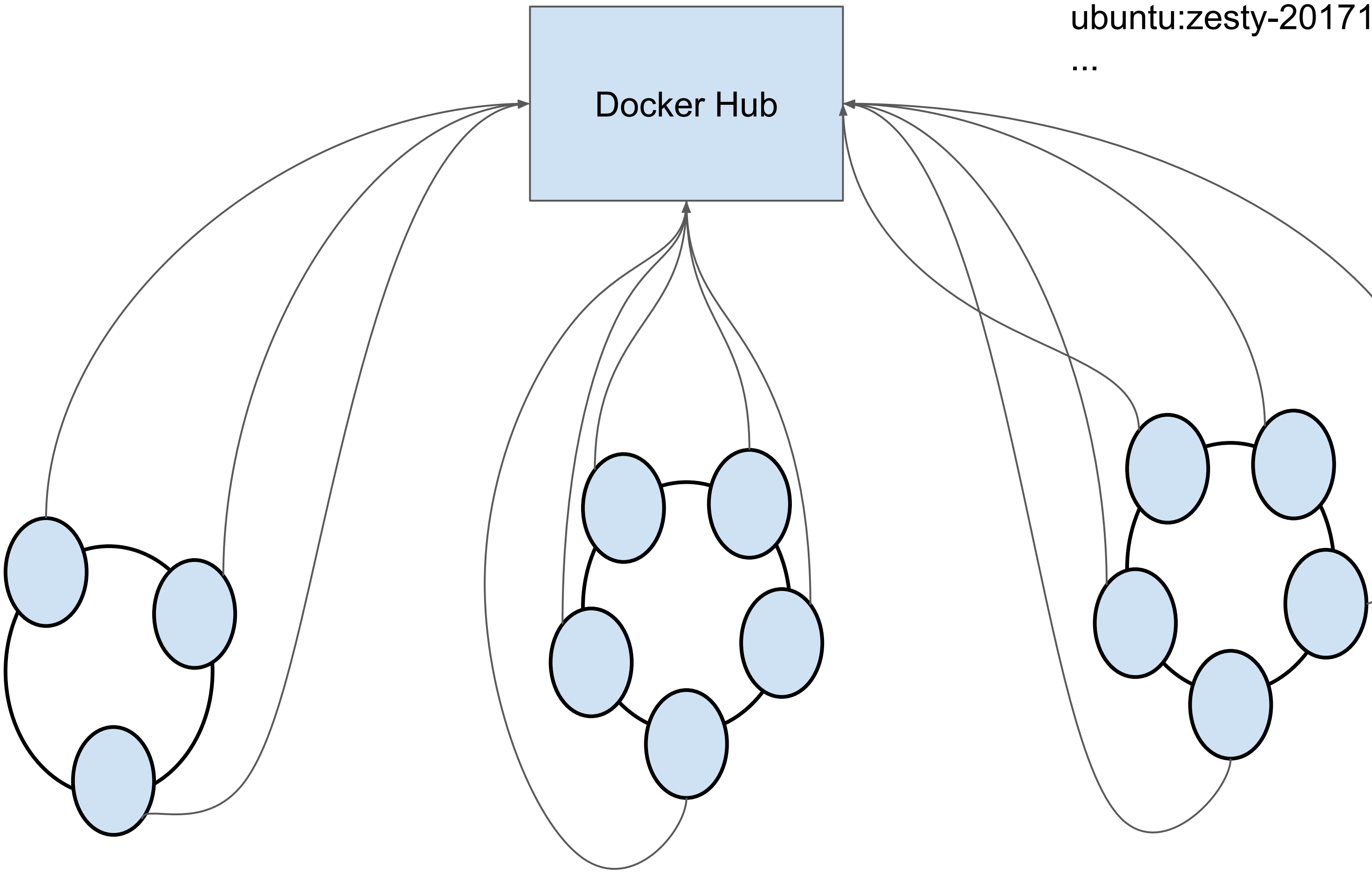
download
the .aci file (tarball)
over HTTP

Kubernetes node

rkt fetch coreos.com/etcd:v3.1.11

the manifest
might contain
other parent images

Wasting network bandwidth



ubuntu:zesty-20171114
ubuntu:zesty-20171117
...

Kubernetes cluster 1

Kubernetes cluster 2

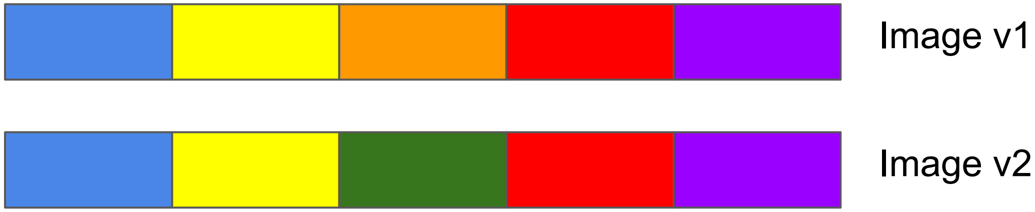
Kubernetes cluster 3

Previous work with Bittorrent

- quayctl
  - https://quay.io/
  - quayctl https://coreos.com/blog/torrent-pulls
- rkt - previous discussions
  - https://github.com/rkt/rkt/issues/405
  - https://github.com/rkt/rkt/issues/798
  - https://github.com/rkt/rkt/issues/1751

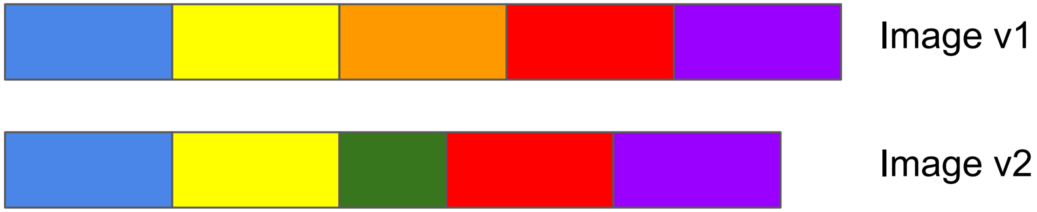
Motivation for casync

- ★ Only download necessary changes between versions



Problem when adding/removing bytes

- ★ Only download necessary changes between versions

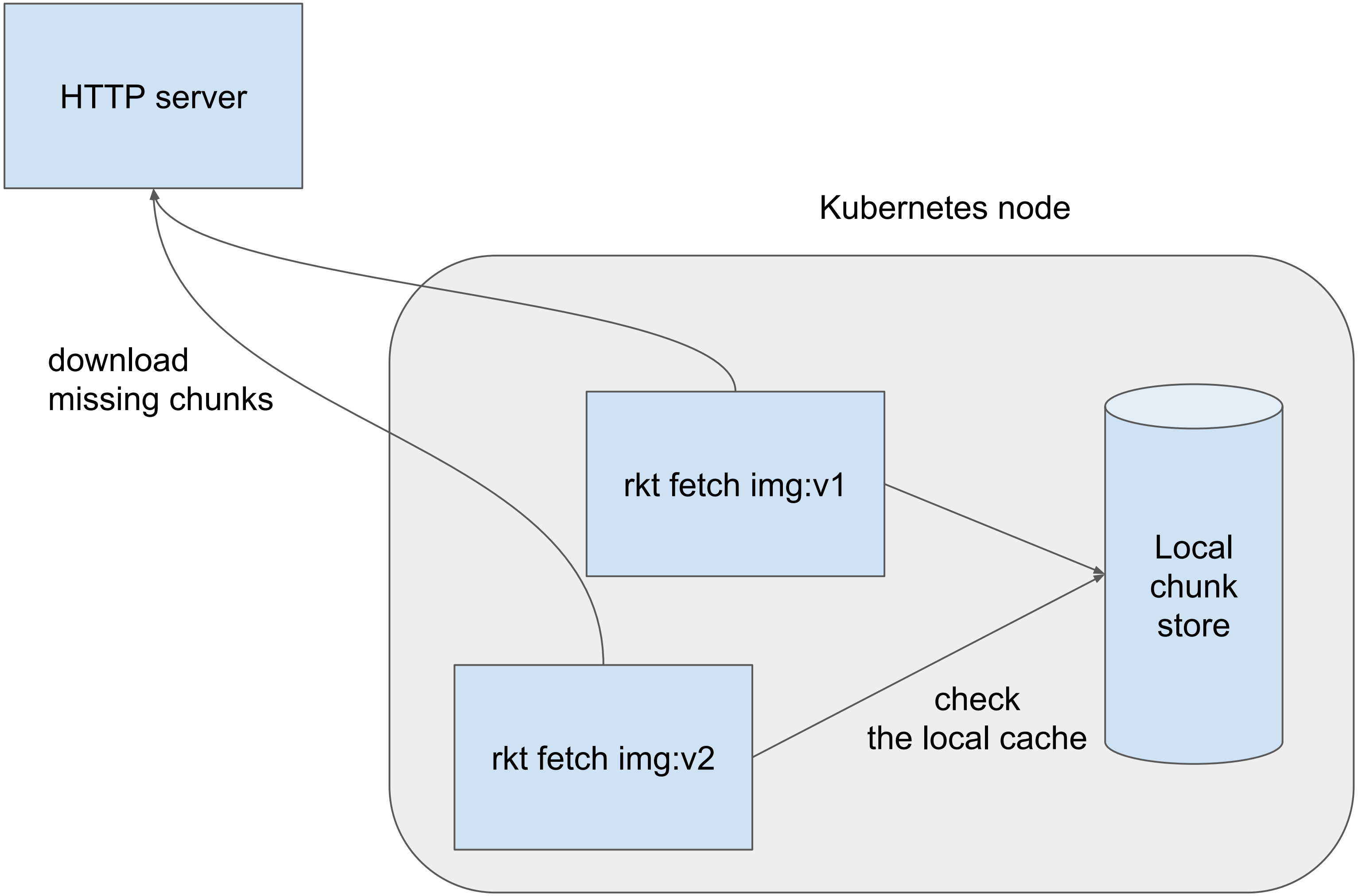


- ★ Chunks of variable size
  - ○ Chunk size based on content

How does casync work?

- https://github.com/systemd/casync
- Steps: building the index file & chunk store
  - Serialization
  - Split the serialization into chunks
  - Hash each chunks
  - Compress & store in the chunk store
- Extracting:
  - Download the index file
  - Download the missing chunks
  - Reverse steps

casync integration with rkt



casync integration with rkt

- Status: just an experiment for now

```
<meta name="ac-discovery"
      content="kinvolk.io/ubuntu http://kinvolk.io/rootfs.caidx">
```

http://kinvolk.io/default.castr/e4a2/${chunk_hash}.cacnk

- rkt branch
  - https://github.com/kinvolk/rkt/tree/alban/casync

TODO

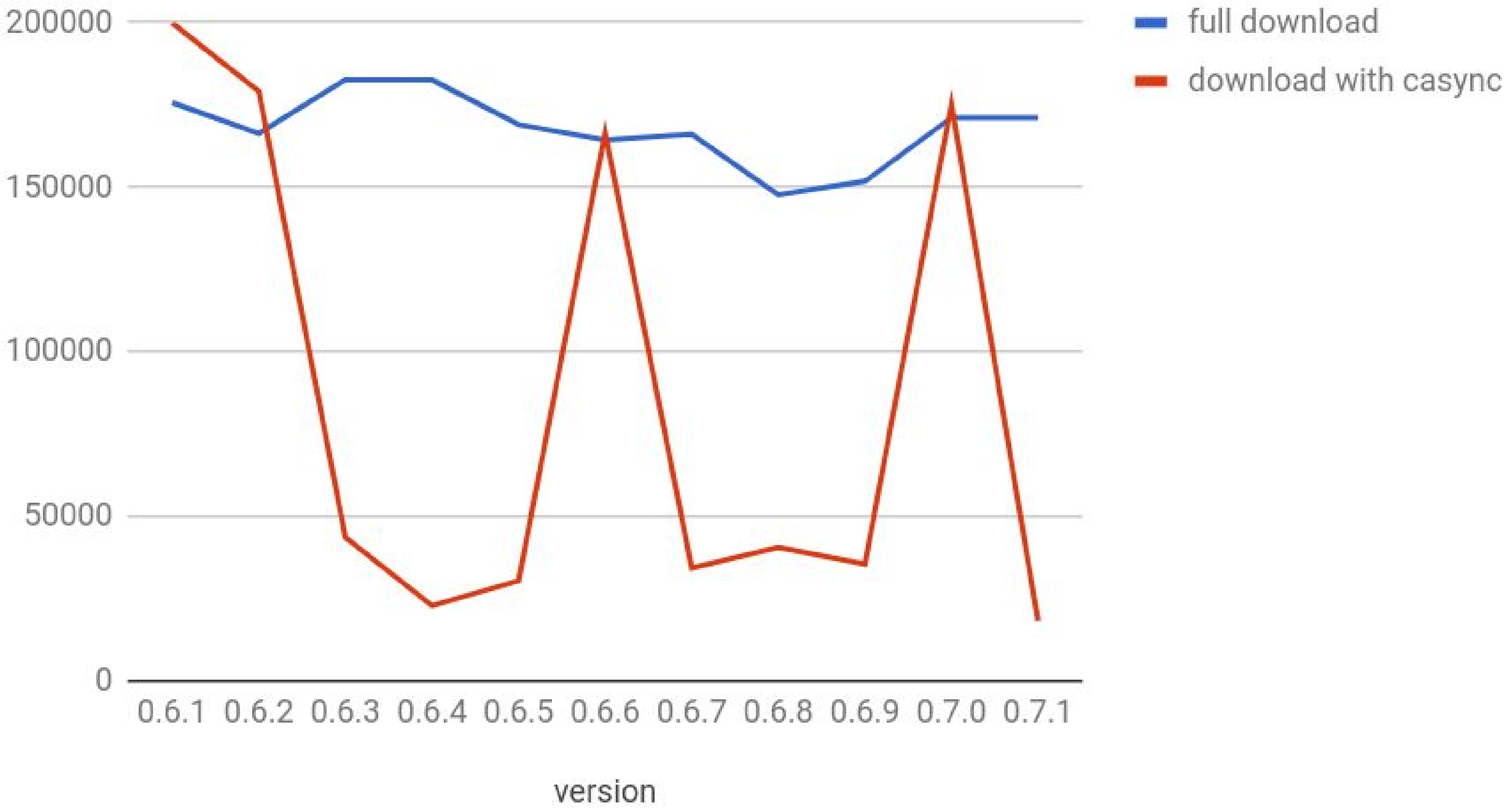
- ★ Try the desync library (in Go)
  - ○ https://github.com/folbricht/desync
- ★ Cache GC
- ★ FUSE: start the container sooner and download on-demand

Experimenting

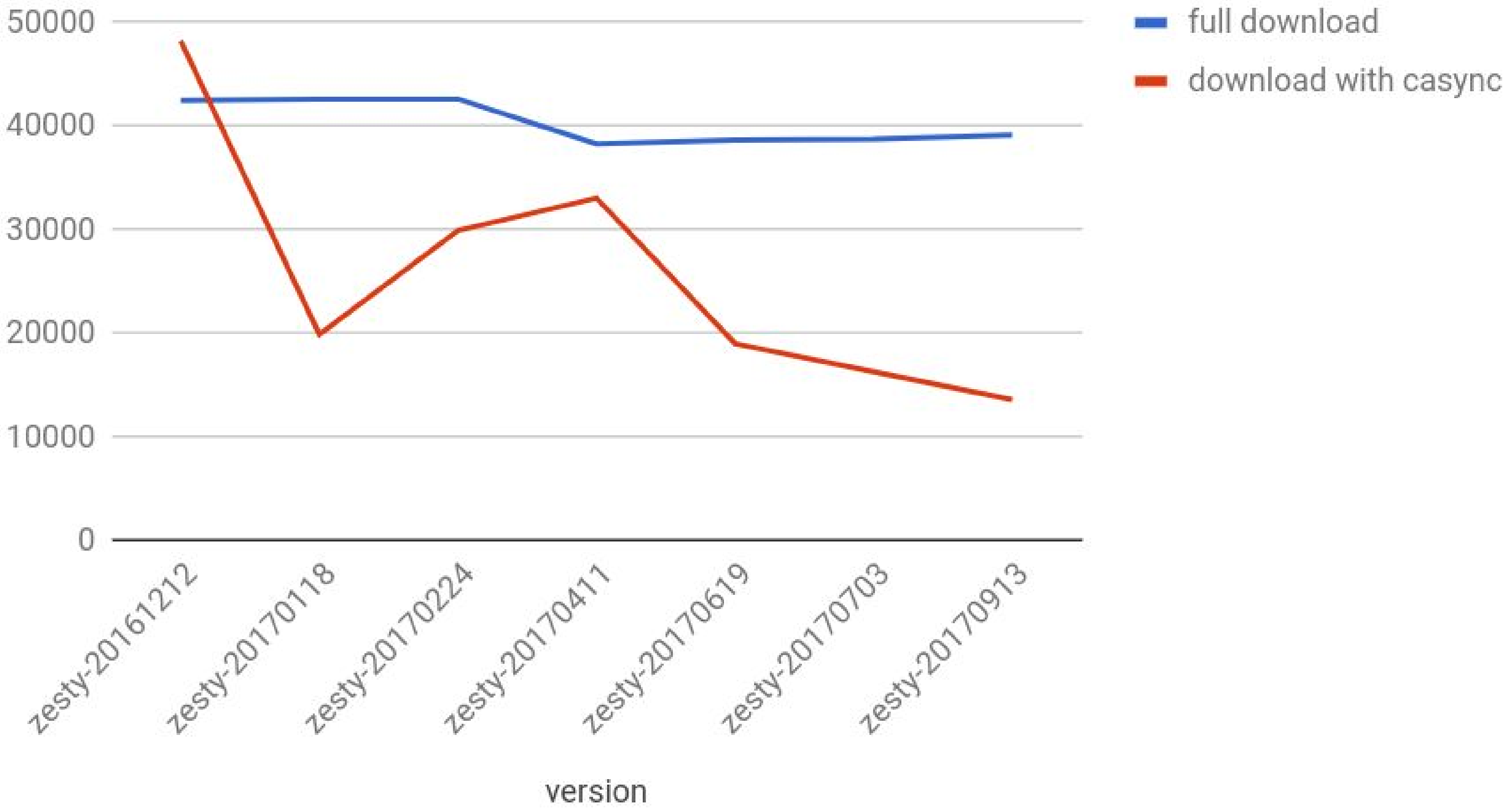
Does this actually save network bandwidth?

https://github.com/kinvolk/casync-measurements

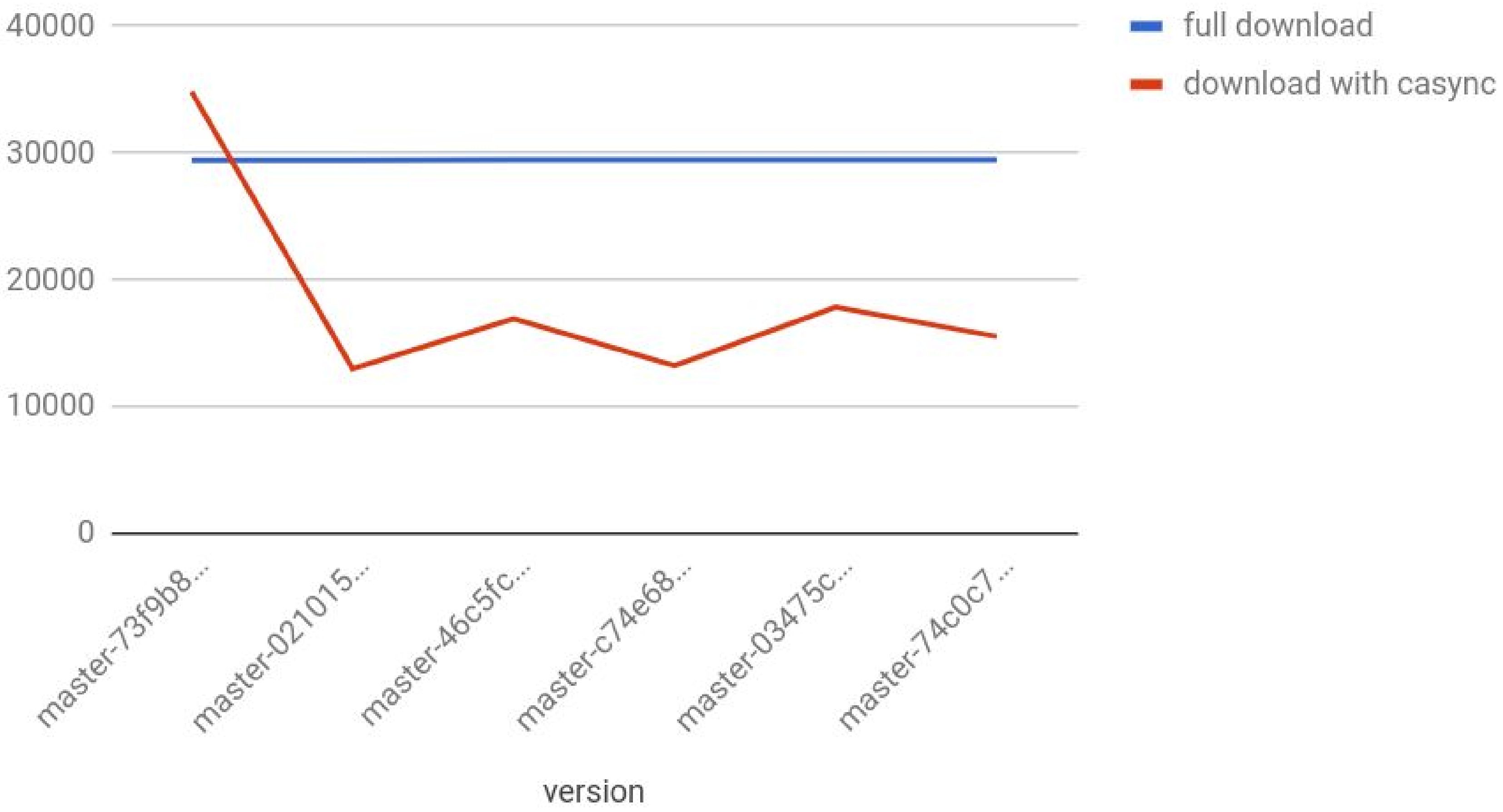
library/registry



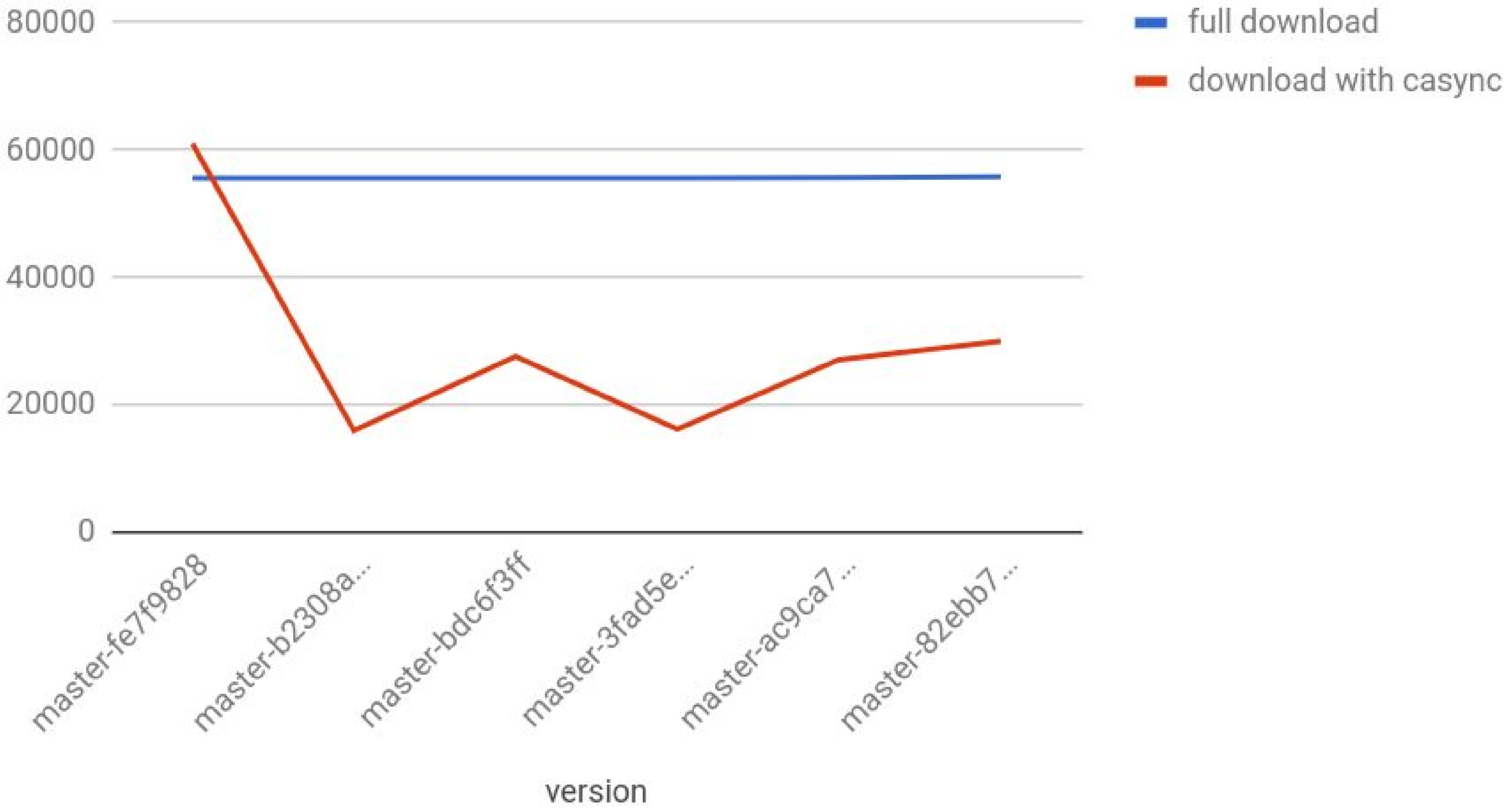
library/ubuntu

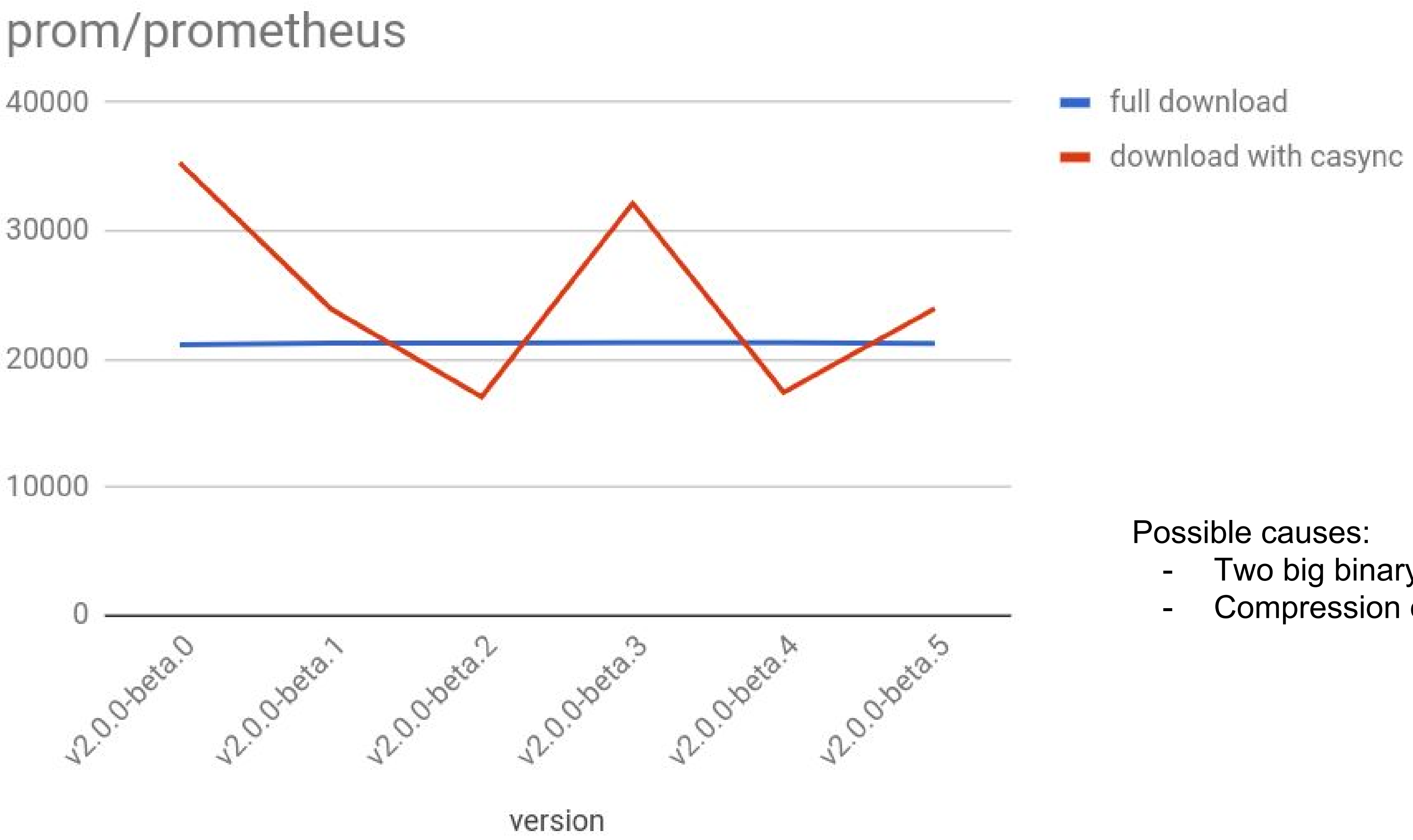


weaveworks/scope



weaveworksdemos/front-end





Possible causes:

- Two big binary files
- Compression on chunks

Conclusion

- We can save significant network bandwidth on some images but not all
- It would require more work