Rapid IP redirection with SDN and NFV

Jeffrey Lai, Qiang Fu, Tim Moors

December 9, 2015

Background

Enabling ISP-CDN collaboration

Our work is mainly in the context of enabling and enhancing ISP-CDN collaboration

Current examples include in-cache style collaborations (Akamai, netflix, google...) and ISP-operated CDNS (Telstra, Comcast, Verizon..)

Recent enablers such as the prominence of SDN and NFV technologies provide a strong motivation to enhance these collaborations

SDN, NFV basics

To keep things short, I'll assume you already know the basics of SDN and NFV

Main important factors: Global knowledge and control (SDN), scalability and flexibility (NFV)

CDN Basics

Content Delivery Networks (CDNs) generally utilise a DNS-based request rerouting scheme

When clients want to access content, they perform a DNS request to associate an IP address to a hostname

These users are typically load balanced/redirected according to geography, server load, etc.

Let's ping google from two different locations!

```
# My VPS in Canada:
root@debian:~ ping -c 2 www.google.com
PING www.google.com (74.125.225.16) 56(84) bytes of data.
64 bytes from ord08s12-in-f16.1e100.net (74.125.225.16): icmp_req=1 ttl=56 time=19.2 ms
64 bytes from ord08s12-in-f16.1e100.net (74.125.225.16): icmp_req=2 ttl=56 time=19.2 ms

# Here at the hilton:
jeff@kanye:~ ping -c 2 www.google.com
PING www.google.com (74.125.224.112) 56(84) bytes of data.
64 bytes from lax02s19-in-f16.1e100.net (74.125.224.112): icmp_req=1 ttl=56 time=13.9 ms
64 bytes from lax02s19-in-f16.1e100.net (74.125.224.112): icmp_req=2 ttl=56 time=14.2 ms
```

Ten minutes later...

Let's ping google from two different locations!

```
# My VPS in Canada:
root@debian:~ ping -c 2 www.google.com
PING www.google.com (173.194.46.112) 56(84) bytes of data.
64 bytes from ord08s13-in-f16.1e100.net (173.194.46.112): icmp_req=1 ttl=56 time=19.4 ms
64 bytes from ord08s13-in-f16.1e100.net (173.194.46.112): icmp_req=2 ttl=56 time=19.1 ms

# Here at the hilton:
jeff@kanye:~ ping -c 2 www.google.com
PING www.google.com (216.58.217.196) 56(84) bytes of data.
64 bytes from lax17s05-in-f4.1e100.net (216.58.217.196): icmp_req=1 ttl=55 time=15.7 ms
64 bytes from lax17s05-in-f4.1e100.net (216.58.217.196): icmp_req=2 ttl=55 time=15.5 ms
```

Client assumptions...

IP-server mappings are unlikely to change

IP-hostname mappings are unlikely to change

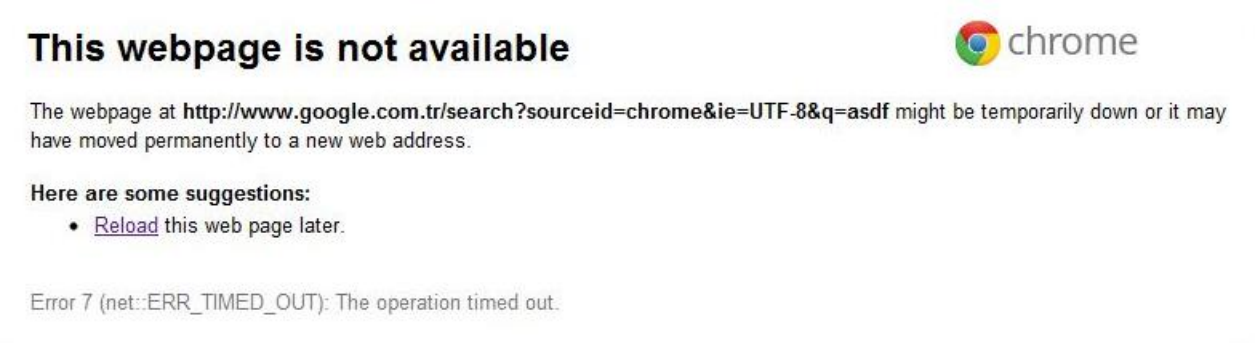
IP-hostname mappings are correct & ideal

...And breaking them

What happens when we break these assumptions?

...And breaking them

What happens when we break these assumptions?



The problem

- DNS caching is a thing!
  - Browsers
    - DNS prefetching
    - Internal caching
  - Operating systems
  - Caching resolvers on the LAN
  - Routers

The problem

DNS is fundamentally pull-based

Network operators can't push latest info to users, must wait for users to request it

Round robin only uses one address at a time

My research

Getting around DNS limitations

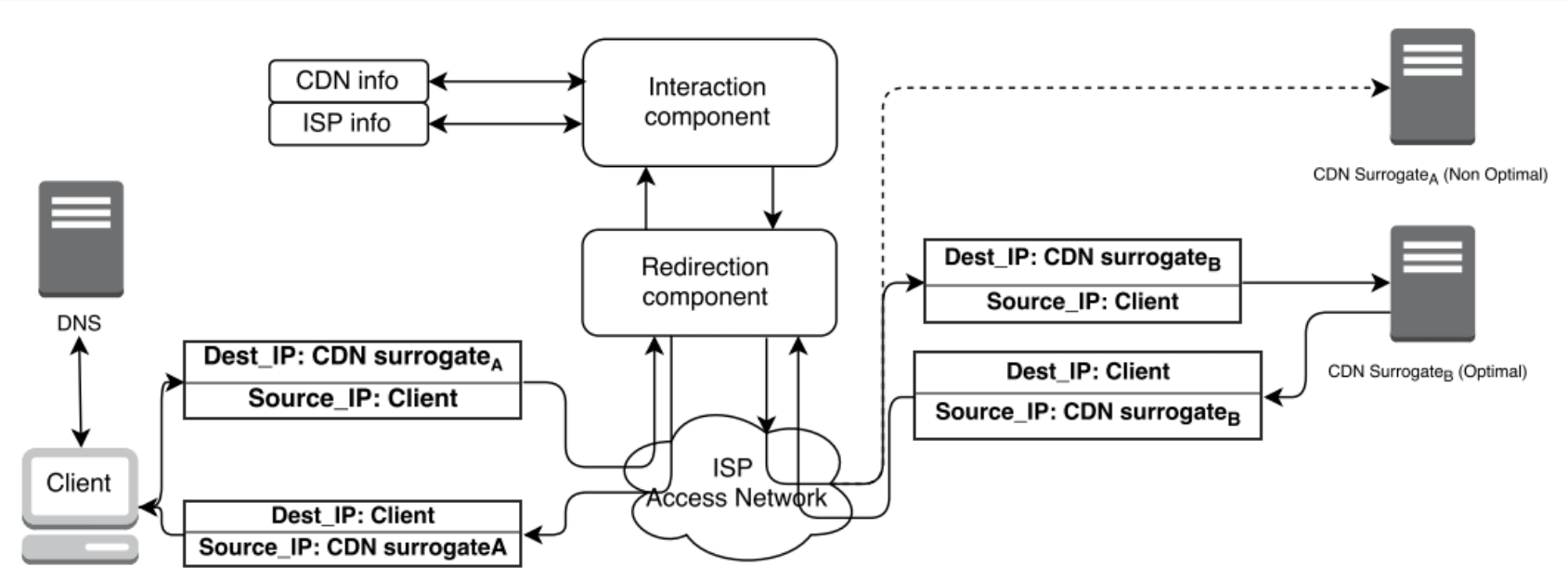
Network operators know the latest server details, users don't

- New server IP addresses
- Network load/status
- Server utilisation

We want to enable/enhance CDN/ISP collborations

Implementing an SDN solution

Why not perform DNAT/SNAT operations on packets passing through the network?



Implementing an SDN solution

Why not perform DNAT/SNAT operations on packets passing through the network?

A naive approach would be to install these rules in an SDN switch, with a separate rule for each TCP flow

But this might be beyond the capacity of current SDN switches[1]

[1] A single host usually opens 6 or more TCP sessions per server, leading to ten or more rules per host, per server!

Sample flow rules

Table miss flow that catches traffic from users (10.0.0.0/24) to the CDN replicas

Priority	In-port	Protocol	IP SRC	IP DST	SRC port	DST port	Actions
10	*	TCP	10.0.0.0/24	10.0.1.0/24	*	80	Send packet to controller

DNAT and SNAT flows to transparently redirect a user to a different CDN server

Priority	In-port	Protocol	IP SRC	IP DST	SRC port	DST port	Actions
20	*	TCP	10.0.0.1	10.0.1.2	4444	80	DNAT DST IP from 10.0.1.2 to 10.0.1.3, send out switch port 3
20	*	TCP	10.0.1.3	10.0.0.1	80	4444	SNAT SRC IP from 10.0.1.3 to 10.0.1.2, send out switch port 1

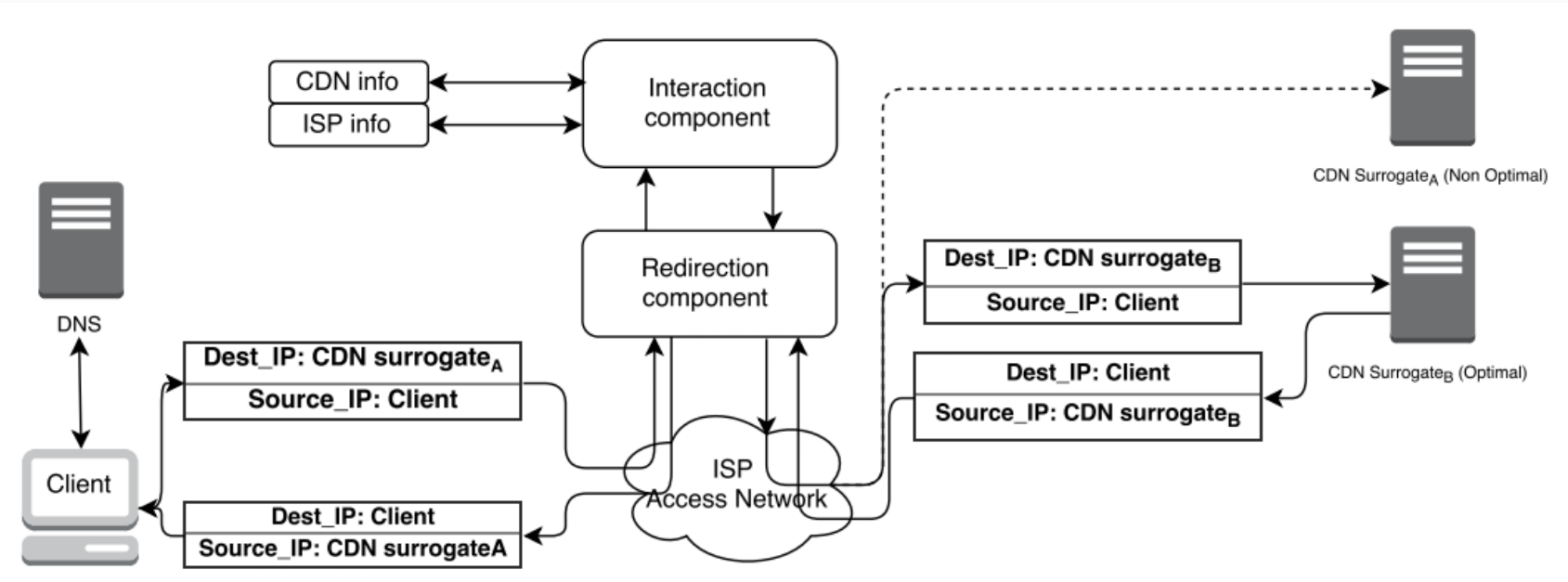
Instead of performing NAT in the switch, why not perform it in a separate device?

We still want to maximise ISP-CDN collaboration, so we also want to bake in custom APIs and functionality

Changing scale requirements as traffic demands change

So why don't we spin up a customized NAT VNF, and perform our operations there?

This way, state information is stored in the VNF and not in the switch, plus we get NFV benefits (Scalability, agility, flexibility).



Note: In this scheme, NAT VNF can be placed pretty much anywhere on the path between client and CDN. We suggest placing it in the same DC as the switch that directs traffic to the NAT VNF.

Sample flow rules

Flow that catches traffic from users (10.0.0.0/24) to the CDN replicas

Priority	In-port	Protocol	IP SRC	IP DST	SRC port	DST port	Actions
10	*	TCP	10.0.0.1/24	10.0.1.0/24	*	80	Send packet to NAT VM

Flow that catches traffic back from the CDN replicas

Priority	In-port	Protocol	IP SRC	IP DST	SRC port	DST port	Actions
10	*	TCP	10.0.1.0/24	10.0.0.1/24	80	*	Send packet to NAT VM

Only two rules need to be placed into SDN switch[2]!

[2] At this point, using an SDN switch isn't even entirely necessary...

Testing procedure

We created a topology similar to the overview in mininet, and used the Click software routing framework to create our NAT VNF

CDN surrogates hosted identical sets of test data

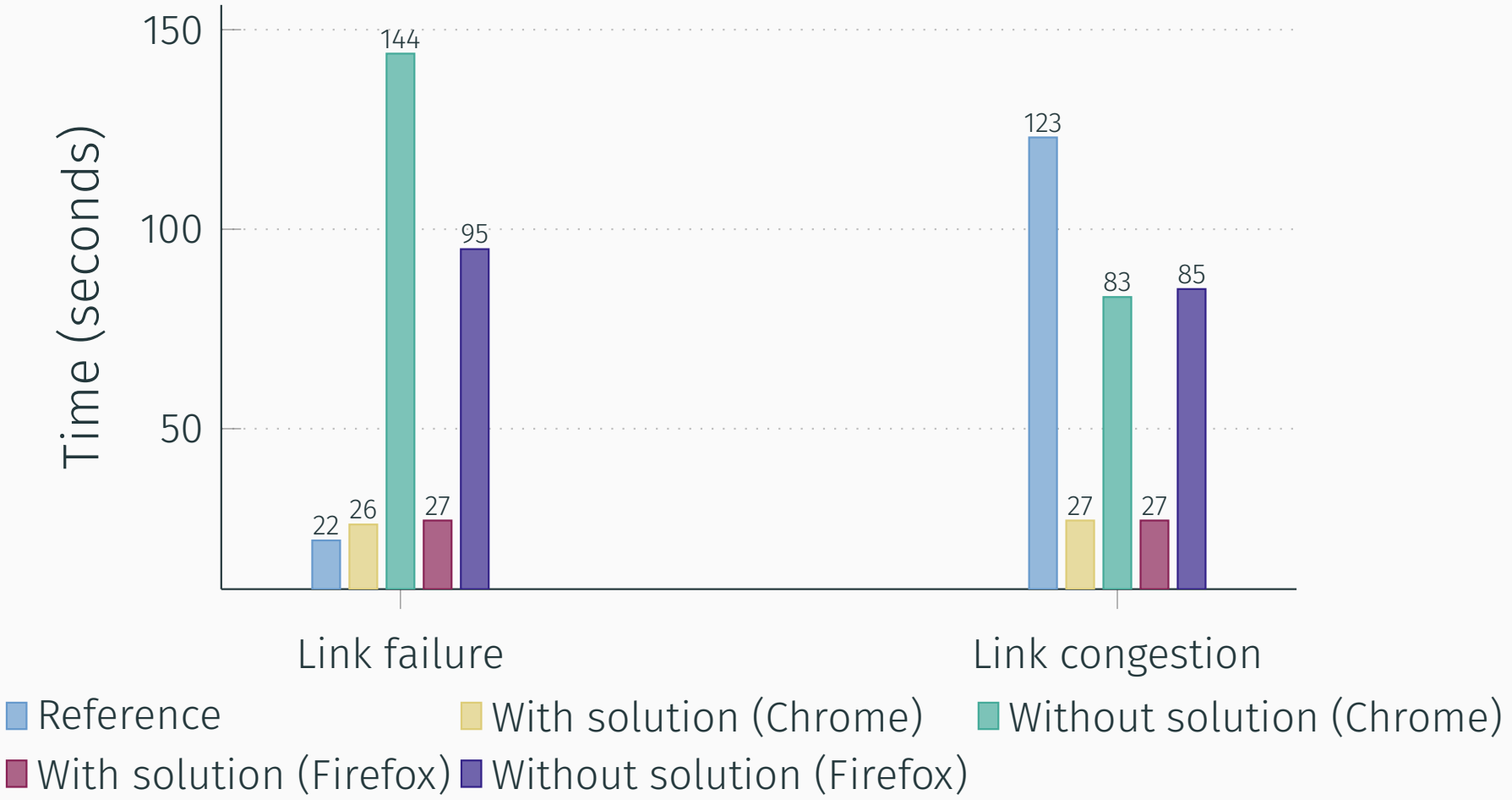
Test data consisted of a series of websites that each contained data: Landing page ->Data set 1 ->Data set 2 ->Data set 3. Each data set contains 11 separate elements (The page manifest, and 10 data elements). An example can be seen at http://jeffl.ai/testpages

An anomaly was injected during this page load sequence (Link failure, link congestion, flash crowd)

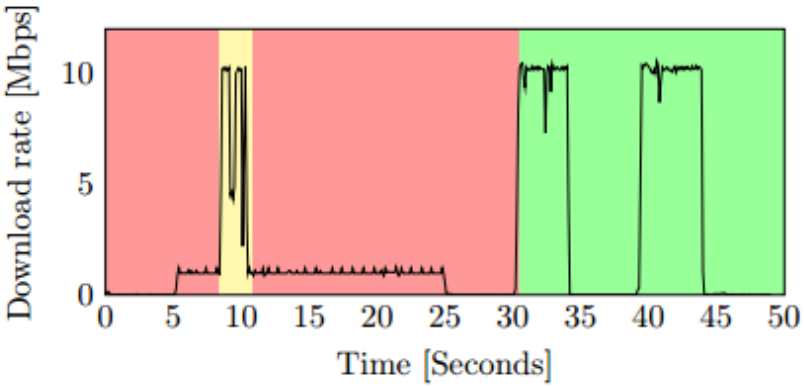
Pages were automated via Javascript

DNS TTL was set to 20 seconds, and browsers were set to have no content caches enabled; HTTP pipelining also disabled.

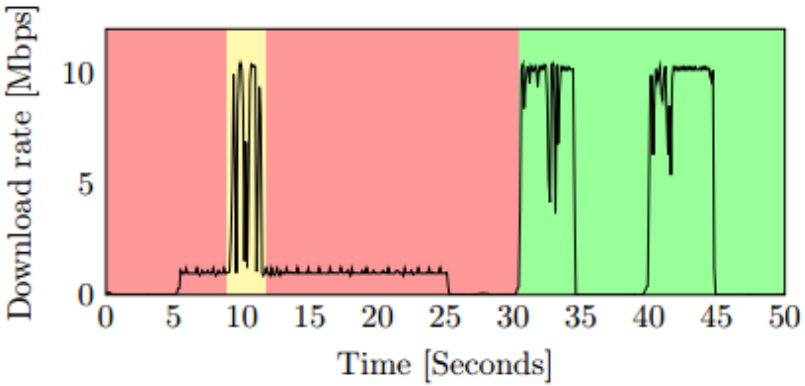
Results


Time (seconds)
150
100
50
144
95
123
83
85
22
26
27
27
27
Link failure
Link congestion
Reference
With solution (Chrome)
Without solution (Chrome)
With solution (Firefox)
Without solution (Firefox)

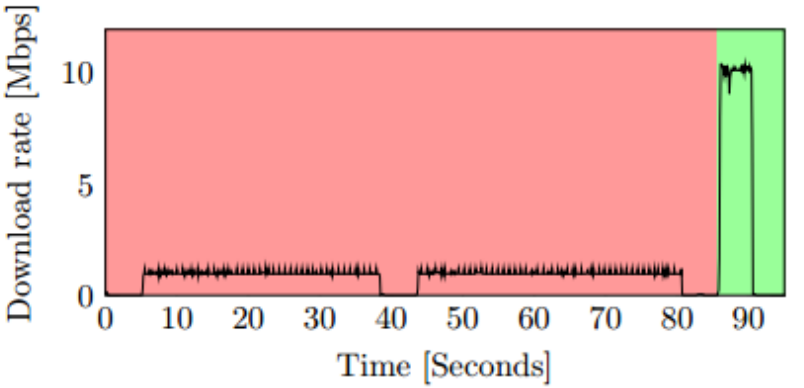
Results, traces for congested link



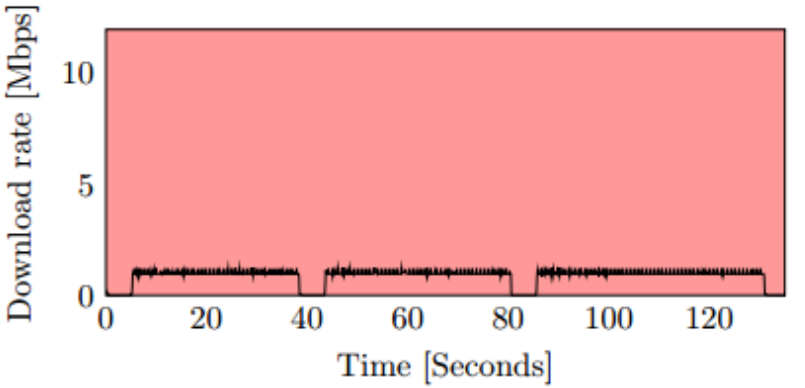
(a) SDN Trace



(b) SDN-NFV Trace



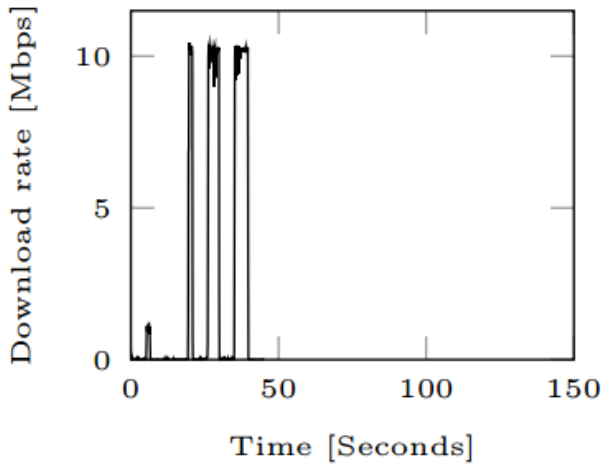
(c) DNS Trace



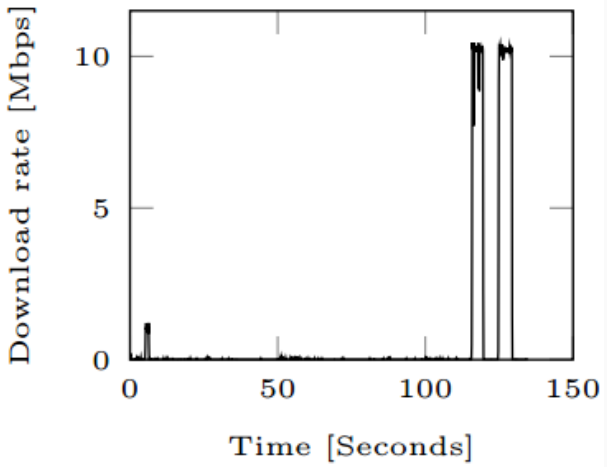
(d) Baseline Trace

These traces & results are from our further work on this project, and have been submitted as part of a journal paper. It uses a slightly different test methodology.

Results, traces for link failure

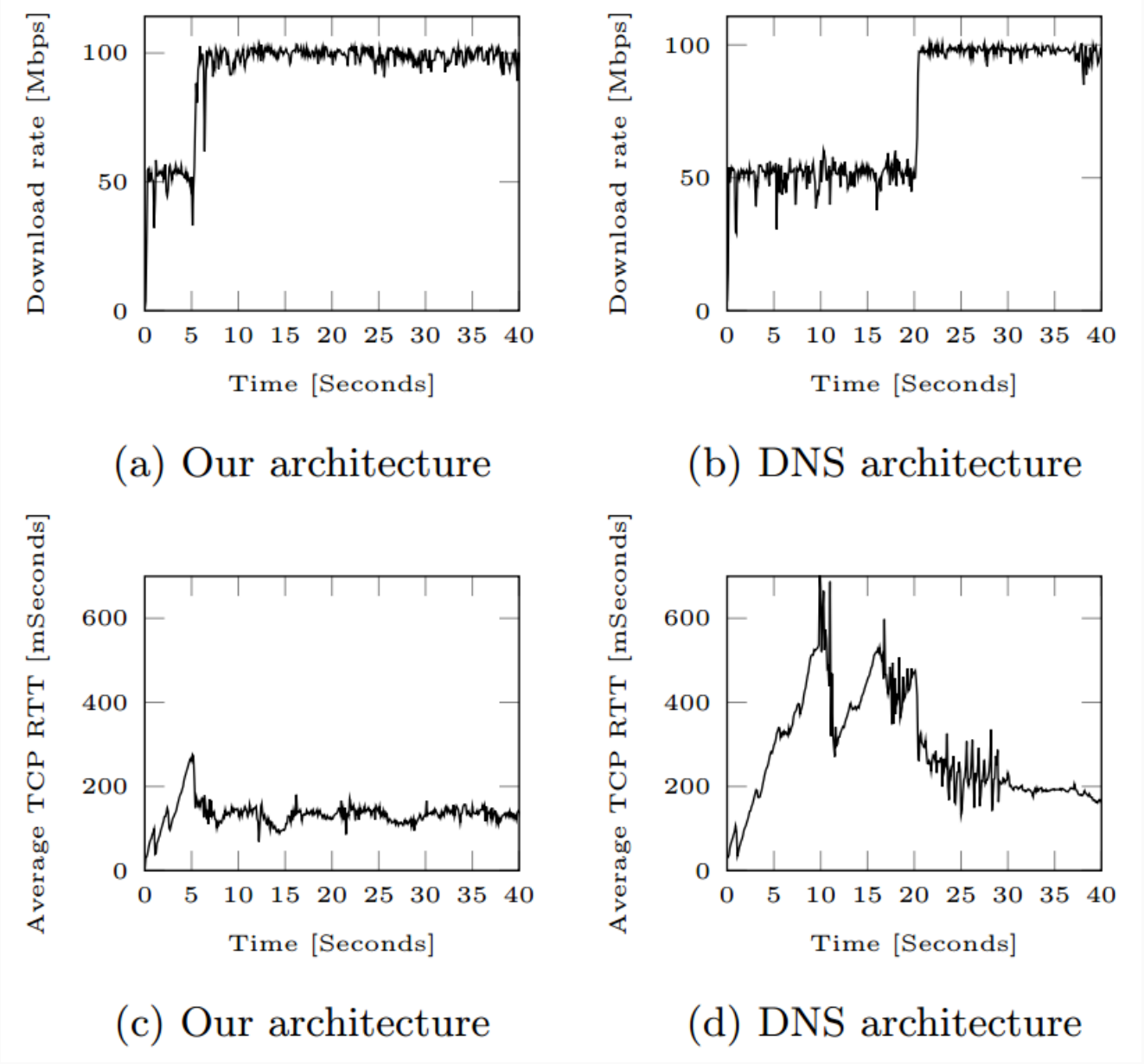


(a) Proposed architecture



(b) DNS architecture

Results, adding extra CDN resources



Conclusions

We have demonstrated a highly flexible architecture for enabling rapid, fine granularity redirection and server assignment of CDN flows

We have performed thorough evaluation of our architecture, showing large gains over the current norm

Also shown that DNS caching can potentially be a serious issue that needs to be addressed for agile networks

Importantly, the architecture we have presented is highly compatible with pre-existing frameworks, allowing for simple rollouts and incremental implementation

Questions?

Feel free to contact me at jeff@jeffl.ai

Slides available at `http://jeffl.ai/globepres.pdf`

Thanks!

What's the scalability like?

SDN-only implementation demonstrably has terrible scaling, potential for many short lived flows.

NFV scheme created to address SDN scalability issue, capability of scaling out - we have developed a consistency verification scheme for this

Bandwidth scalability can be a concern since all traffic from host to CDN passes via NAT VNF. Currently addressing this in future work with a 'fake' anycast methodology

What about anycast?

Currently researching a ‘fake’ anycast implementation, promising results.

Native anycast has flapping/management/control issues, can’t control behavior past controlled networks

What about HTTP redirect and similar methods?

Increased RTT, time to first byte.

Still prone to DNS issues

What about non-HTTP applications?

What about CDN strategies?

Our system is agnostic to CDN surrogate selection strategies!

The inputs/outputs our system cares about are IP endpoints, and network conditions; same as current CDN surrogate selection techniques.

What about different topologies?

Our system *should* be agnostic to changes in topology.

We perform redirection based on endpoint server IP addresses, so as to rely on an ISP's preexisting IP routing framework.

Currently deploying a testbed in Amazon AWS, planetlab