Maygh: Building a CDN from client web browsers

Liang Zhang, Fangfei Zhou, Alan Mislove, Ravi Sundaram

The problem

Serving web content at massive scale:

- operator supplies all infrastructure and network bandwith
- spread distribution costs among the users
- use content distribution networks

General idea

Allow end users to help web site operator distribute static objects

Maygh

automatically build CDN from visiting users’ unmodified web browsers!

Approach 1: plugin/standalone

research systems:

- Firecoral
- BuddyWeb
- Flower-CDN
- Web2Peer

Akamai

NetSession

Approach 1: drawbacks

Limited reach:

AdBlock - 144 million users (<5% total)

Akamai Netsession: 24 million installs

- operators still need to pay Akamai
- is this a virus?

Maygh: features

- no client-side involvement
- minor modifications to the operators's site
- can be deployed today

Maygh: potential

Content Type	% Requests	% Bytes	% Cacheable
Image	70.5	40.3	85.7
JavaScript	13.1	29.0	84.8
HTML	10.7	19.9	30.1
CSS	3.5	8.7	86.5
Flash	0.9	1.3	96.0
Other	1.3	1.0	45.7
Overall	100	100	74.2

Table 1. Breakdown of browsing trace from the top 100 Alexa web sites. Cacheable refers to the fraction of bytes that are cacheable according to the HTTP headers.

Building blocks: content storage

Content storage:

- Indexed Database API - Javascript based object-oriented database
- WebStorage - persistent key-value storage in web browser

Building blocks: direct communication

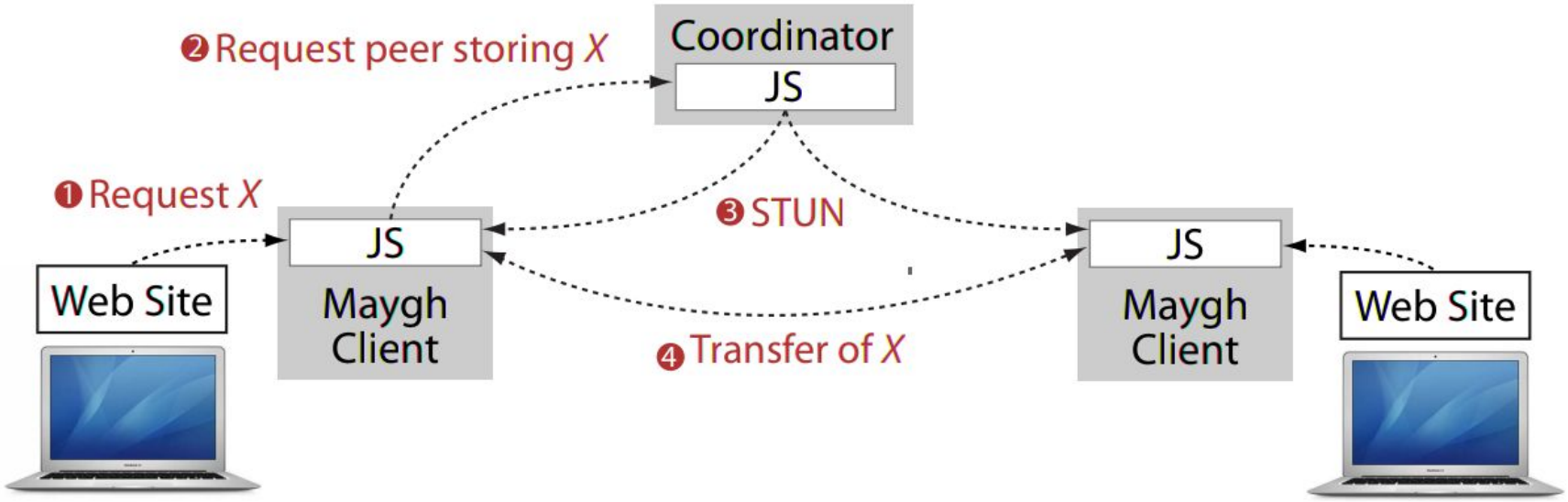
- **RTMFP -** Real Time Media Flow Protocol
  - originating from Adobe Flash, now opened up
  - p2p, but designed only for direct end to end user communication
- **WebRTC** - Web Real-Time Communications
  - API defined by W3C
  - voice calling, video chat, p2p file sharing without need of any plugins!
  - widely adopted: >chrome 23, >firefox 22, >opera 18, >edge 21, mobile browsers too!
  - cannot be disabled on desktop chrome

Design: assumptions

Assumptions:

- content is always available from the operator
- content is named by its content hash (like SHA-256)

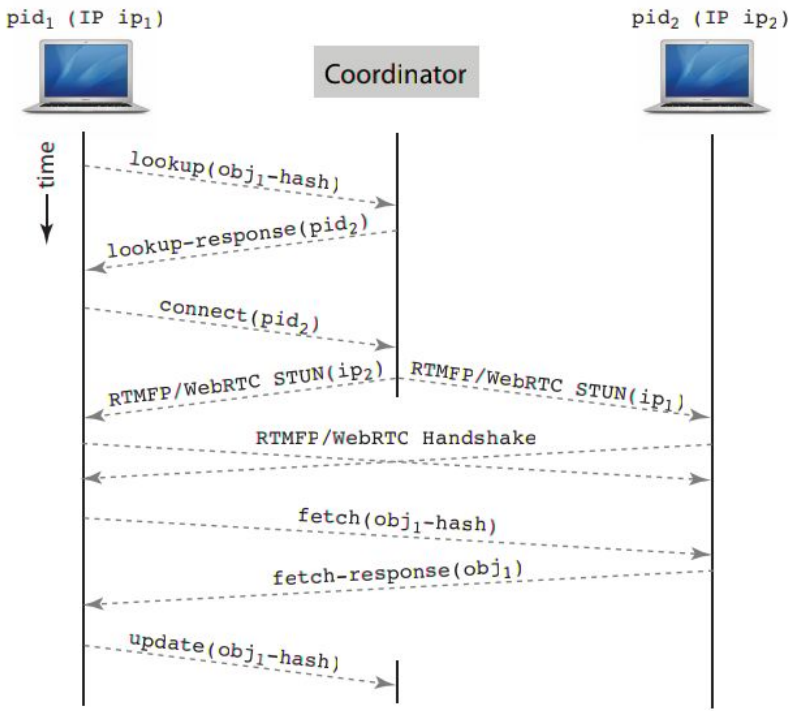
Design


❷ Request peer storing X
Coordinator
JS
❶ Request X
❸ STUN
JS
Web Site
Maygh
Client
JS
Maygh
Client
Web Site
❹ Transfer of X

Client <-> Coordinator

1. **Initializing**: after web page loaded: client does handshake with coordinator, informs about local content, then connection is kept-alive
2. **Content request:** lookup the coordinator, coordinator returns the client (already connected/closest)
3. **Connect to another client:** directly/with a STUN (if behind a NAT)
4. **New content stored:** update coordinator with new objects stored/objects deleted

Client <-> Client



```
<img id=”-id-” src=”-src”>
```



```
<img id=”-id-”>
<script type=”text/javascript”>
      maygh.load(“-hash-”, “-id-”);
</script>
```

Client

Javascript included to the web page (with RMTFP also invisible Flash object conducting communication)

API:

- connect(coordinator)
- load(content_hash, id)

- compatible with AJAX
  - but requires modification of loading logic
- loading object in parallel
- single connection to site’s coordinator, regardless of number of tabs

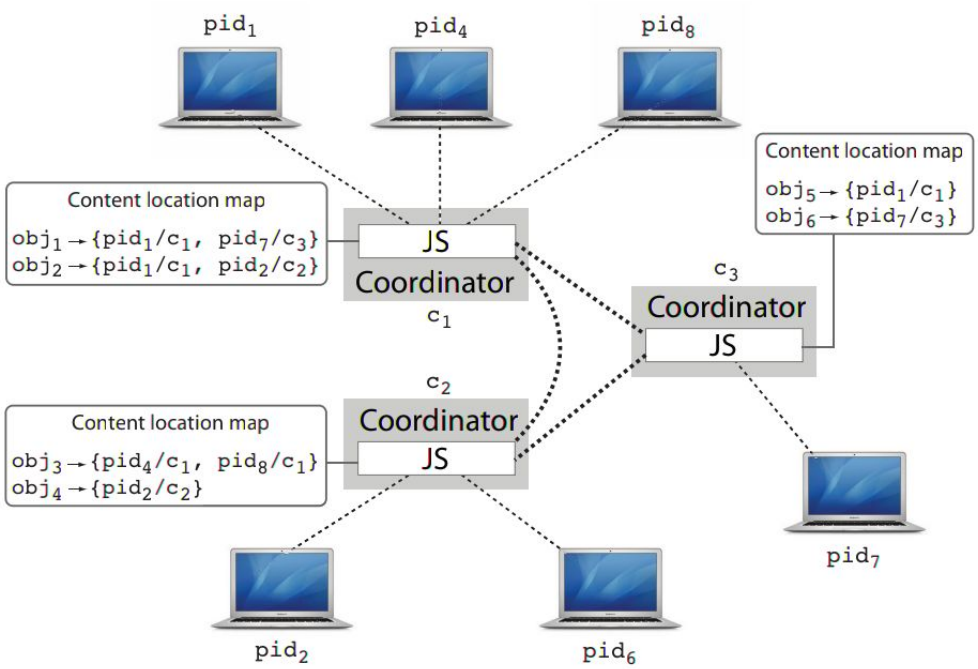
Coordinator

Maintains:

- content location map: content-hash -> list of peer-ids
- client map: peer-id -> list of content-hashes
- number of content bytes down/uploaded per client over period of time
- global upperbound: ratio of upload-download
- limit upload data

Multiple coordinators

Enable clients connected to different coordinators to exchange content!



Multiple coordinators

- content location map: distributed across coordinators
  - coordinator is responsible for storing the content-location map for which its coordinator-id is closest to content-hash key
  - for each peer-id store his coordinator-id
- Lookup: client's coordinator determines coordinator storing the entry in content-location map
- Fetch: client's coordinator contacts appropriate remote's client coordinator to allow direct connection

At most two messages as a response to lookup results in linear scalability.

Security

- forgery: self-certifying content - verification by computing the hash
- DDoS: block accounts, IPs, subnets

Privacy

- coordinator tracks the content stored in user's browser
- clients receive information about other users *views* of content (e.g. IP)
  - disabling Maygh for sensitive content
  - client-side privacy controls
  - background requests to random pieces of content (cover traffic)
- determining specific client's cache is difficult
  - some client closest to the requester is returned, randomized
  - restricting access: coordinator authenticating requests
  - restricting access: client needs to know the content-id

Impact on users

- lowering the costs
  - reducing the need for placing ads!
  - Maygh as opt-in: when enabled, less ads displayed
- asymmetric bandwith
  - tests showing no significant impact on the system

Mobile users

- user sessions much shorter than on desktops
- limited battery life
- data access charges

Evaluation

- impact on web clients: increased latency/network overhead?
- scalability: requests per second
- impact on operator: how much Maygh reduces the traffic
- field tests

Implementation

- coordinator
  - RMTFP,
  - single-threaded, event-driven
  - Node.js
- client
  - 4.2 KB of javascript + 2.6KB Flash object

Tests: console based client

Client-side microbenchmarks

Loaded to		Loaded from LAN BOSTON	Loaded from Cable BOSTON	Loaded from DSL NEW ORLEANS
LAN	BOSTON	229 / 87	618 / 307	1314 / 707
Cable	BOSTON	771 / 283	702 / 314	1600 / 837

Table 2. Average time (ms) to load first / second 50 KB objects using Maygh with RTMFP.

Loaded to		Loaded from LAN BOSTON	Loaded from Cable BOSTON	Loaded from DSL NEW ORLEANS
LAN	BOSTON	72 / 16	364 / 120	544 / 354
Cable	BOSTON	284 / 57	577 / 107	765 / 379

Table 3. Average time (ms) to load first / second 50 KB objects using Maygh with our proof-of-concept WebRTC implementation.

Coordinator scalability

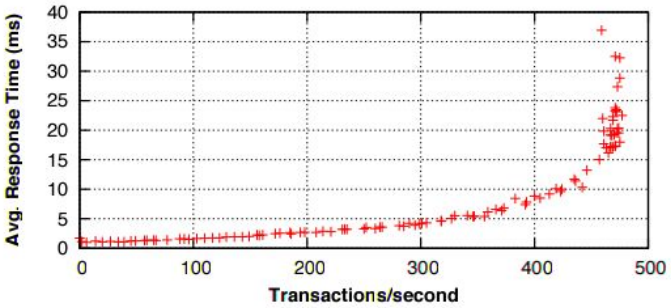


Figure 4. Average response time versus transaction rate for a single coordinator. The coordinator can support 454 transactions per second with under 15ms latency.

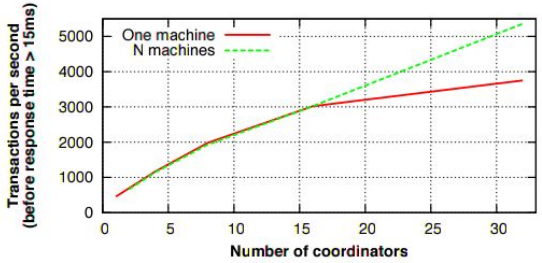


Figure 5. Average response time versus request rate for a multiple coordinators working in tandem, in two different placements of the coordinators across machines. Close-to-linear scaling is observed as more coordinator nodes are added (the one-machine set of coordinators show lower performance after 16 coordinators are placed on a single machine due to the effects of hyperthreading).

Etsy's trace-based simulation

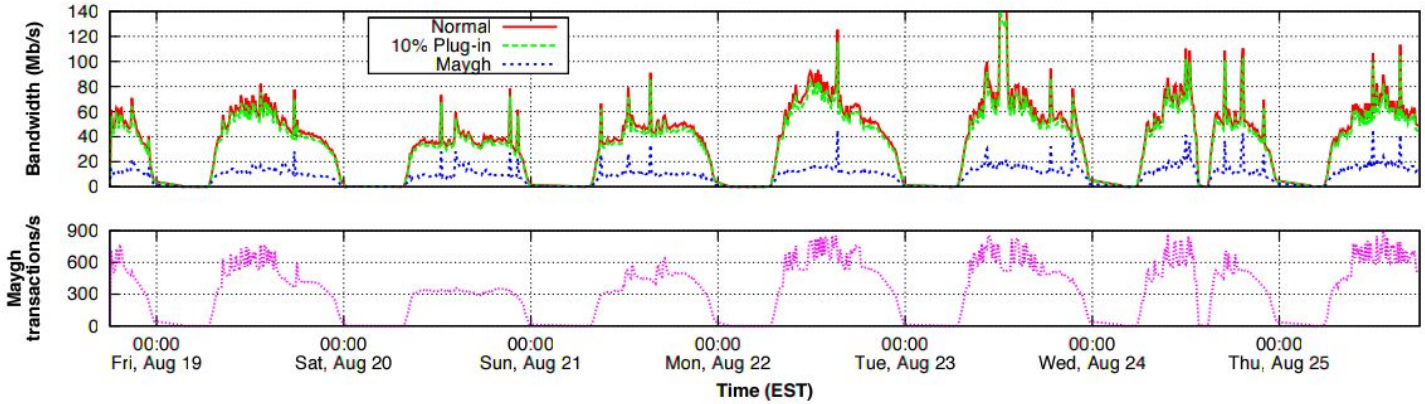


Figure 6. TOP: Bandwidth required at the operator, as normal (with no plug-ins or Maygh), with a 10% plug-in deployment, and with Maygh; a 10% plug-in deployment results in 7.8% bandwidth savings, while Maygh results in 75% bandwidth savings. BOTTOM: Request rate observed at the Maygh coordinator; the rate is almost always below 800 requests per second, and is easily handled by a four-core server.

Etsy's trace-based simulation

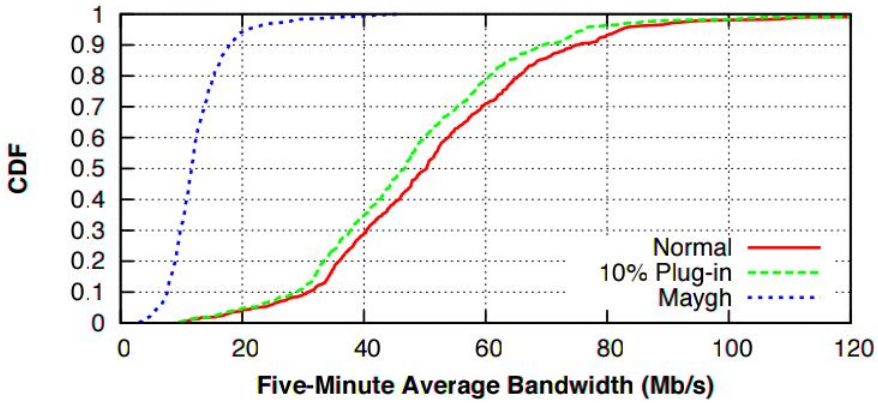


Figure 7. Cumulative distribution of five-minute average bandwidth required at the operator for the same trace as Figure 6.

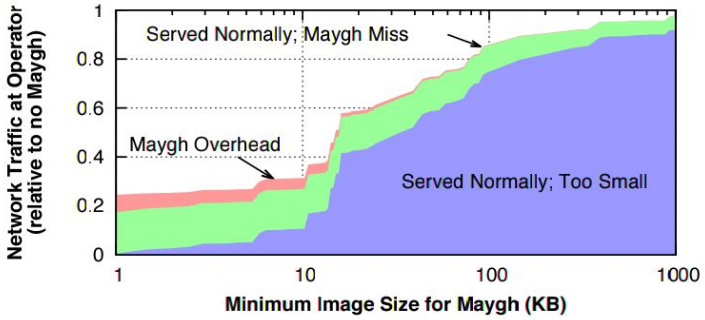


Figure 8. Network traffic at the operator, relative to the trace without Maygh, for different minimum Maygh image sizes. The traffic consists of three components: images that are served normally because they are too small, images that are served normally because Maygh cannot find an online client able to serve it, and overhead caused by Maygh (downloading the client code and protocol overhead). If Maygh serves all images, the operator would experience 75% less traffic due to images.

Etsy's trace-based simulation

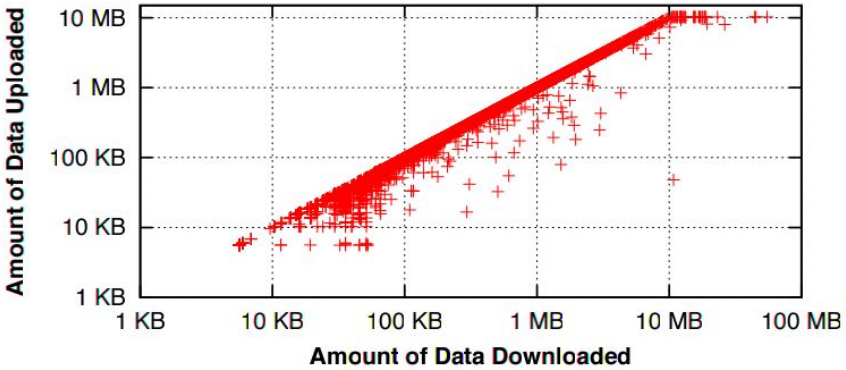


Figure 9. Network traffic at the clients, comparing amount of data uploaded to downloaded. We observe that the client workload distribution of Maygh is "fair", meaning most clients are requested to upload an amount of data that is proportional to the amount they download.

Thank you!

https://github.com/leoliangzhang/maygh