Maygh: Building a CDN from client web browsers

Liang Zhang  Fangfei Zhou  Alan Mislove  Ravi Sundaram
Northeastern University

EuroSys ’13, Prague
Content exchange and the Web

**Web is popular mechanism** for content distribution
- News sites, content sharing, movies

**Web is fundamentally client-server**
- I.e., Web site operator serves every client

Popular Web sites receive millions of hits per day
- Need to handle a large number of requests

How do large, popular web sites distribute content?
Distributing web content

Options for content distribution:

1. Serve on your own
   Purchase machines, network bandwidth

2. Pay content distribution networks (CDNs)
   Akamai, Limelight, Clearway, ...

3. Rent cloud services
   Amazon EC2, Azure, App Engine...

In all cases, significant monetary burden on web site operator
How do operators pay?

Operators typically use two models to support site:

1. User subscriptions (e.g., Netflix, New York Times, Rdio)
   Limited user base

2. Advertising (e.g., YouTube, Yahoo, Google*)
   Resort to data-mining user data, privacy implications

Few choices limit set of sites that can exist
Free web sites have to accept advertising

Can we give web site operators another option?
Idea: Clients help distribute content

Typical properties of popular web sites:
- Many users
- Same content viewed by many users
- Content are largely static

Insight: Recruit web clients to help serve content

Technically challenging
- Significant user churn
- Web has client–server architecture

But, we are not the first to explore this idea...
Alternate Approaches

1. Browser plugins
   FireCoral, SwarmPlugin

2. Client-side software
   Akamai’s NetSession, PPLive

Both require installation of additional software
   Typically with few incentives
   E.g., Adblock Plus, most popular plug-in: 4.2% installations

Can we build a system that does not require additional software?
Goal: Build content distribution system for the Web
Allow web browsers to assist in content distribution to other users

Requirements:
Works with today’s web sites, browsers
No client side changes

Maygh
Serves as a cache for static web content
Takes advantage of recent HTML5 browser features
Significantly reduces bandwidth requires for operator

Result: On-demand CDN built from web browsers
Outline

1. Motivation
2. Maygh design
3. Security and privacy implications
4. Evaluation
Maygh design overview

Maygh: Drop-in content distribution system
Serves as a distributed cache
Assume content always available from origin

Maygh serves static content
E.g., image, CSS, JavaScript
Content must be named by content-hash

Key challenge: Browsers not designed to communicate directly
Browsers distinct from Web servers
Use new techniques to allow browser to serve content
Protocol: RTMFP or WebRTC

Two peer-to-peer protocols for Web browsers
Designed for *direct audio/video chats*
Both support NAT traversal via STUN

Adobe Flash RTMFP
Supported in *Flash player 10.0 since 2008*
Available in 99% of browsers

WebRTC
W3C standard, *actively under development*
Currently in Firefox and Chrome
Maygh overview
Maygh overview

Etsy

Maygh Coordinator

Alice

Bob
Maygh overview

- Etsy
- Maygh Coordinator

Alice → Maygh Coordinator → Bob

Alice and Bob are connected to the Maygh Coordinator, which is connected to Etsy.
Introduce a middlebox: Maygh Coordinator
Run by website operators

Serves two purposes:
1. Serves as a directory for content
   Keeps track of content in user’s browsers
   Content-hash -> {set of online clients}

2. Allows browsers to establish direct connections
   Supports NAT traversal using STUN with RTMFP/WebRTC

Techniques to allow multiple coordinators in paper
Can scale to support high churn, 1000s requests/second
Client-side changes

Implement Maygh client-side library in Javascript
Add it to the site’s pages

Browsers use RTMFP/WebRTC to communicate with coordinator
Allows bi-directional communication
Online client is always connected to coordinator

Use LocalStorage to storage browsed content
Persistent cache, up to 5MB/site
Easily programmatically accessed

Insert downloaded objects in LocalStorage
Treat like LRU cache
How does an operator use Maygh?

Web site operators need to do three things:

1. Run coordinator(s)

2. Include Maygh Javascript
   
   ```html
   <script src="maygh.js">
   ```

3. Change mechanism for loading content

   ```html
   <img id="pic-id" src="http://www.foo.com/...">
   ```
   replaced with

   ```html
   <script>
   maygh.load("pic-hash", "pic-id");
   </script>
   ```
Outline

1. Motivation
2. Maygh design
3. Security and privacy implications
4. Evaluation
Security

Can users serve forged content?
   Can detect forged content using content-hash

Can users violate the Maygh protocol?
   E.g., claim to have content, DoS attacks

Use similar techniques that are in-use today
   Block accounts, IP address, or subnets
   Existing defenses against DDoS

Fairness
   Operator controls coordinator, choice of uploading peer
   Maygh tracks content users upload/download
   E.g., Ensure no user has contributes more resources than they use
Privacy

Can users view content they are not allowed to?
- Content secured by its hash
- **Naming content implies access**
  Similar semantics to Flickr, other sites today

Can users figure out what others have browsed?
- Client **receive information about views**
  Can use cover traffic, pre-fetch requests
  Or, allow user to disable Maygh for certain content

Privacy implications similar to other Hybrid-CDN models
- NFL’s p2p streaming, FireCoral, PPLive
Outline

1. Motivation
2. Maygh design
3. Security and privacy implications
4. Evaluation
Evaluation overview

Implemented **Maygh using RTMFP**
Full browser support today, easy to get user base
Also built proof-of-concept WebRTC client

Includes both Maygh coordinator and client-side library
Client: 657 lines of Javascript, 214 lines of ActionScript
Coordinator: 2,944 lines of Javascript

Code open-source, available at

http://github.com/leoliangzhang/maygh
### How much additional latency?

<table>
<thead>
<tr>
<th>Accessed from</th>
<th>Served from</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Maygh</td>
</tr>
<tr>
<td>LAN (Boston)</td>
<td><strong>229 / 87 ms</strong></td>
</tr>
<tr>
<td>Cable (Boston)</td>
<td><strong>618 / 307 ms</strong></td>
</tr>
<tr>
<td>DSL (New Orl.)</td>
<td><strong>1314 / 707 ms</strong></td>
</tr>
<tr>
<td>Cable (Boston)</td>
<td><strong>771 / 283 ms</strong></td>
</tr>
<tr>
<td></td>
<td><strong>702 / 314 ms</strong></td>
</tr>
<tr>
<td></td>
<td><strong>1600 / 837 ms</strong></td>
</tr>
</tbody>
</table>

**Flash RTMFP and WebRTC proof-of-concept implementations**

**Fetch 50 KB objects from other peer**
Show First/Subsequent object loading time

**Overall, latency is sufficient for many Web sites**
Can also be hidden using pre-fetching techniques
### How much additional latency?

<table>
<thead>
<tr>
<th>Accessed from</th>
<th>Served from</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LAN (Boston)</td>
<td>Cable (Boston)</td>
<td>DSL (New Orl.)</td>
</tr>
<tr>
<td>LAN (Boston)</td>
<td>229 / 87 ms</td>
<td>618 / 307 ms</td>
<td>1314 / 707 ms</td>
</tr>
<tr>
<td></td>
<td>72 / 16 ms</td>
<td>364 / 120 ms</td>
<td>544 / 354 ms</td>
</tr>
<tr>
<td>Cable (Boston)</td>
<td>771 / 283 ms</td>
<td>702 / 314 ms</td>
<td>1600 / 837 ms</td>
</tr>
<tr>
<td></td>
<td>284 / 57 ms</td>
<td>577 / 107 ms</td>
<td>765 / 379 ms</td>
</tr>
</tbody>
</table>

Flash RTMFP and WebRTC proof-of-concept implementations

Fetch **50 KB objects** from other peer
Show First/Subsequent object loading time

Overall, latency is sufficient for many Web sites
Can also be **hidden using pre-fetching techniques**
How much bandwidth can Maygh save?

Deploying Maygh to large website is challenging
   Instead, perform simulation

Use 1-week anonymized Akamai access logs from Etsy
   Top-50 US web site, online marketplace
   205M requests, 5.7M IPs
   2.77TB total network traffic

85% of Etsy’s bandwidth is static images

Simulation setup
   Client stay on page for 10 to 30 seconds
   Ensure fairness
      Clients never upload more than downloaded, or more than 10 MB
How much bandwidth can Maygh save?

Median bandwidth used drops
From 50.3 Mb/s to 11.7 Mb/s (a 77% drop)
Even with significant churn

75% reduction in 95th-percentile bandwidth
Only requires one 4-core coordinator
How much bandwidth can Maygh save?

Median bandwidth used drops
From 50.3 Mb/s to 11.7 Mb/s (a 77% drop)
Even with significant churn

75% reduction in 95th-percentile bandwidth
Only requires one 4-core coordinator
How much bandwidth can Maygh save?

Median bandwidth used drops
From 50.3 Mb/s to 11.7 Mb/s (a 77% drop)
Even with significant churn

75% reduction in 95th-percentile bandwidth
Only requires one 4-core coordinator
Real-world deployment

Set up special version of our department’s web server
Set up coordinator within our department

Invite graduate students
18 users for 3 days
Total of 374 photos viewed, 24% served from other Maygh client
Lower than simulation because more users on Etsy

Take-away:  Compatible with today’s
  Browsers (Firefox, Safari, Chrome)
  Websites
Summary

Substantial monetary burden to host popular Web site
Site operators typically resort to advertising to pay bills

Idea: Recruit web clients to help distribute content
Without requiring any additional client-side software

Maygh
Serves as cache for static Web content
Operator runs coordinator, allows clients to communicate

Evaluation demonstrated practicality, efficacy
Open-source and available to research community
Questions?

http://github.com/leoliangzhang/maygh
Cacheable Web content

Dynamically generated web pages popular

So, how much content is static, cacheable?
I.e., what is the potential for system like Maygh?

Conduct a experiment
Consider top 100 websites from Alexa’s ranking
Simulate web browsing via random walk of five pages per site
Consider content with Cache-Control: public cacheable

Result:
On average, 74.2% of bytes are cacheable
Maygh could serve a significant fraction of bytes
### Potential cacheable content

<table>
<thead>
<tr>
<th>Content Type</th>
<th>% Requests</th>
<th>% Bytes</th>
<th>% Cacheable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>70.5</td>
<td>40.3</td>
<td>85.7</td>
</tr>
<tr>
<td>JavaScript</td>
<td>13.1</td>
<td>29.0</td>
<td>84.8</td>
</tr>
<tr>
<td>HTML</td>
<td>10.7</td>
<td>19.9</td>
<td>30.1</td>
</tr>
<tr>
<td>CSS</td>
<td>3.5</td>
<td>8.7</td>
<td>86.5</td>
</tr>
<tr>
<td>Flash</td>
<td>0.9</td>
<td>1.3</td>
<td>96.0</td>
</tr>
<tr>
<td>Other</td>
<td>1.3</td>
<td>1.0</td>
<td>45.7</td>
</tr>
<tr>
<td>Overall</td>
<td>100</td>
<td>100</td>
<td>74.2</td>
</tr>
</tbody>
</table>

Breakdown of browsing trace from the top 100 Alexa web sites.

74.2% of the bytes requested are marked as cacheable. Most static content like images, videos, and SWF are still cacheable.
Scalability of Maygh coordinators?

Single coordinator
- Dual 8-core 2.67 GHz Intel Xeon E5-2670 processors
- 454 transactions per second with under 15 ms latency

More details in the paper
Scalability of multiple coordinators

Multiple coordinators on
- Single machines, using multiple cores (with hyperthreading)
- Multiple machines, using only one core

Close-to-linear scaling
- Single machine performance decreases after 16 coordinators
  - Due to hyperthreading
- A single machine with 4 CPU cores can support Etsy workload