# facebook

(c) 2010 Facebook, Inc. or its licensors. "Facebook" is a registered trademark of Facebook, Inc.. All rights reserved. 1.0

#### facebook

## Ajaxifying, Caching and Pipelining Facebook Website

Changhao Jiang

The Velocity Conference Dec 7th, 2010, Beijing, China

### The beginning - 2004





home search global social net invite fag logout

Puget Sound

Brian Moore's Profile

My Profile [ edit ]

My Friends

quick search

My Groups

My Parties

My Messages

My Account

My Privacy



Picture



Send Brian a Message

Poke Him!

Connection

You are in a relationship with Brian.

Information

Account Info:

Name: Brian Moore

Member Since: May 21, 2005 Last Update: July 19, 2005

Basic Info:

School: Puget Sound '09

Status: Student

Male Sex:

Residence: Todd 311

Birthday: 09/02/1986

Shorewood, WI 53211 Home Town:

High School: Shorewood Hi '05

Contact Info:

bmoore@ups.edu Email:

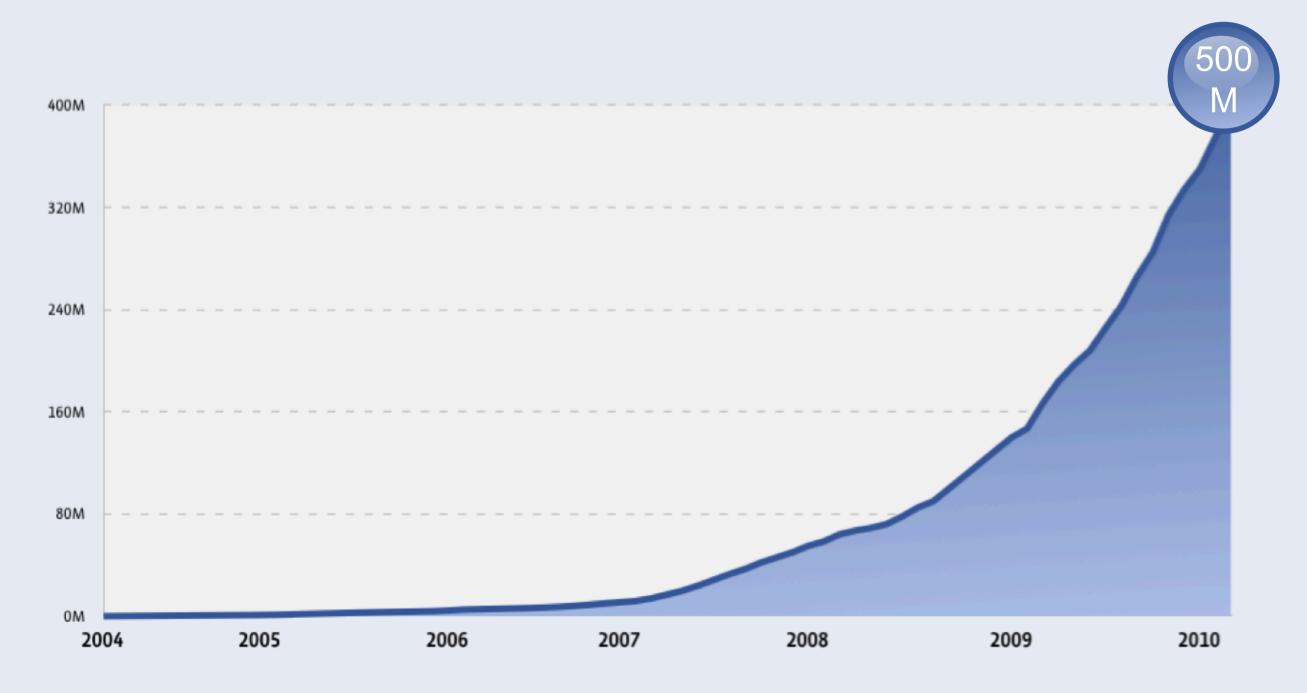
#### Today



## Mission

Give people the power to share and make the world more open and connected

#### 500M Users

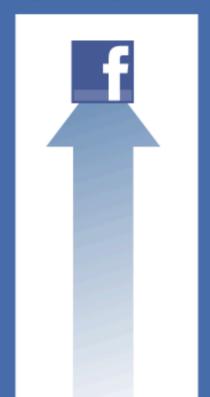


## Today's reality



#### facebook

5hr 25min



## Average hours per month per user!

2hr 17min

















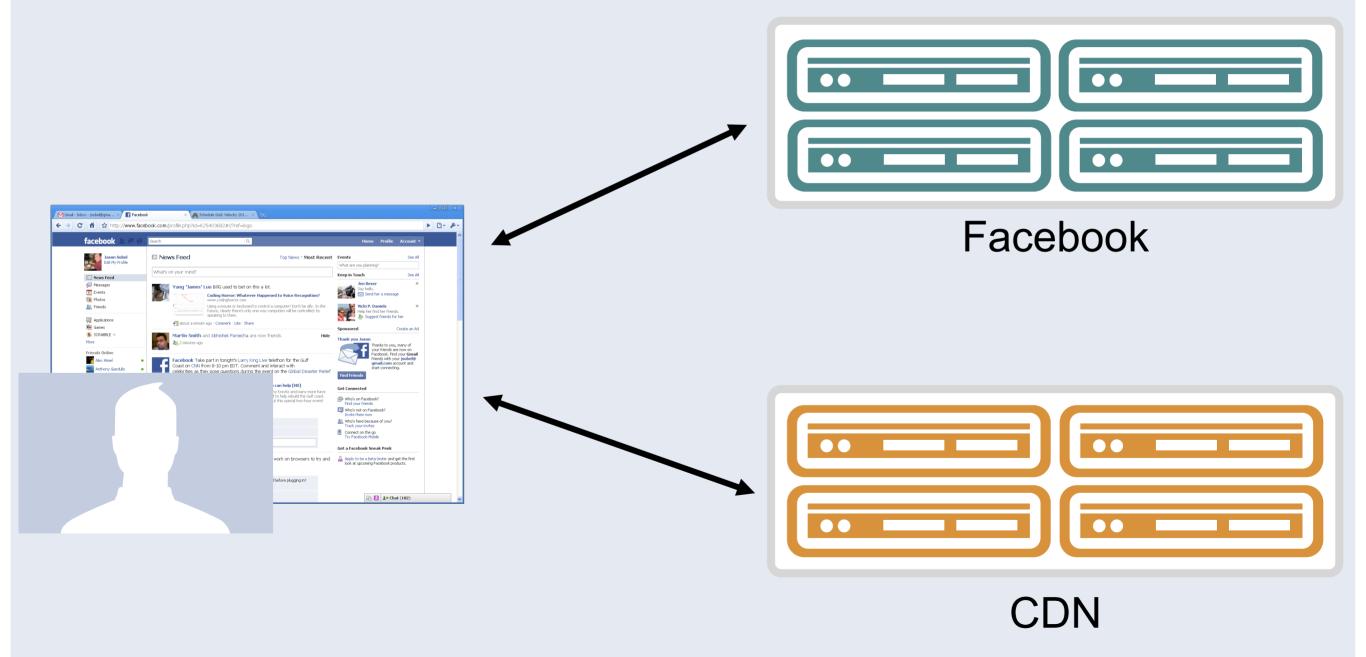






## Site speed is critical to Facebook!

#### What do we measure?



#### What do we measure?

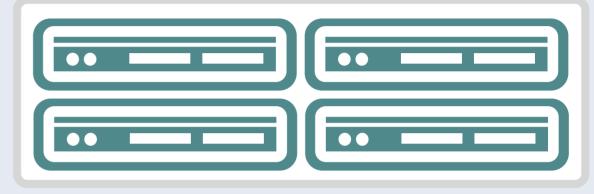
#### server time



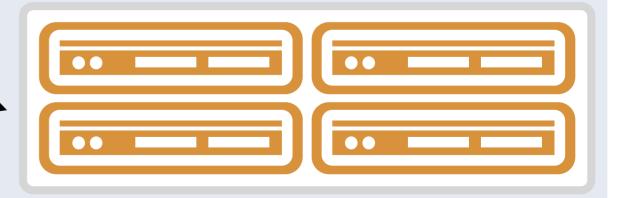






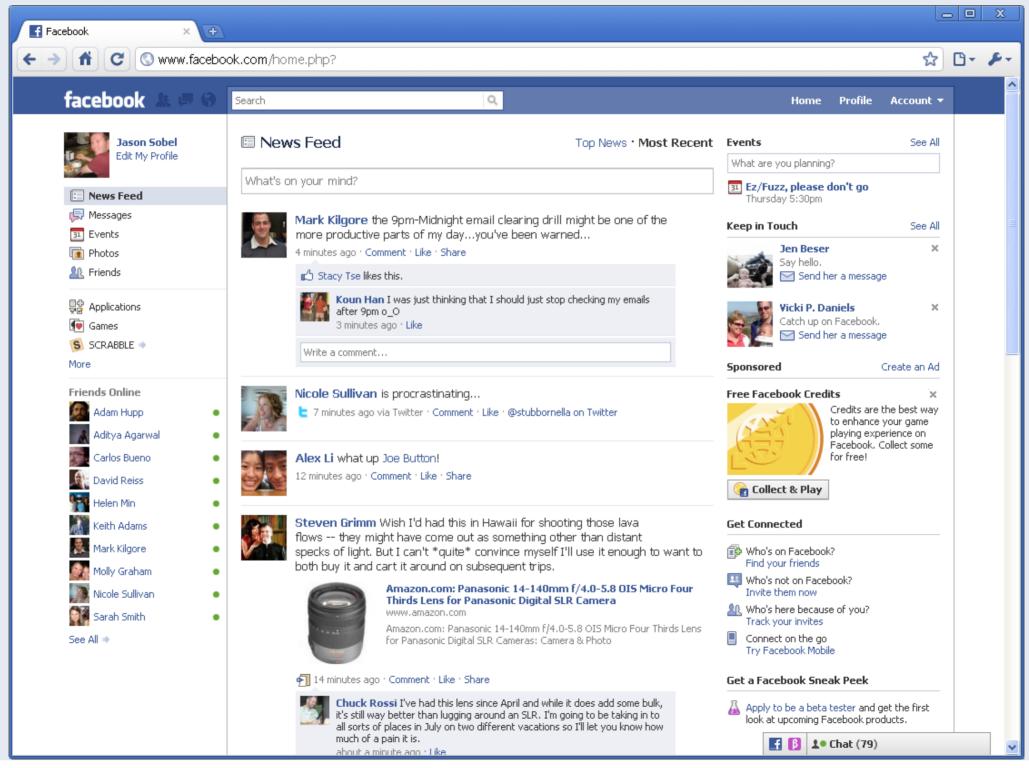


Facebook

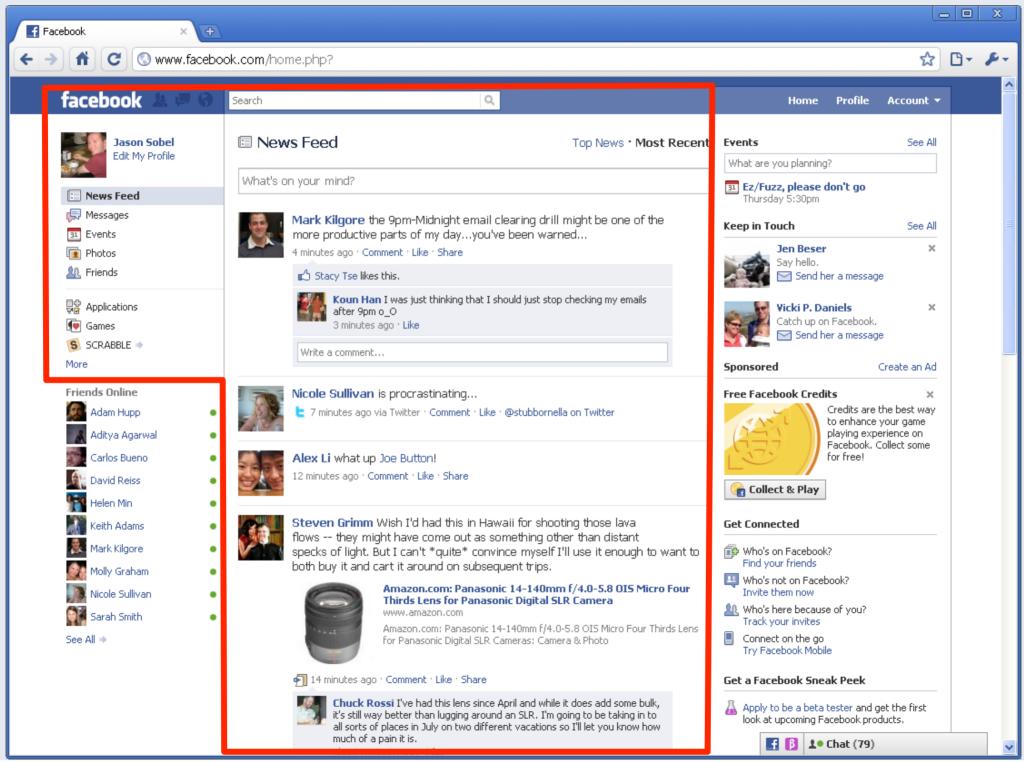


**CDN** 

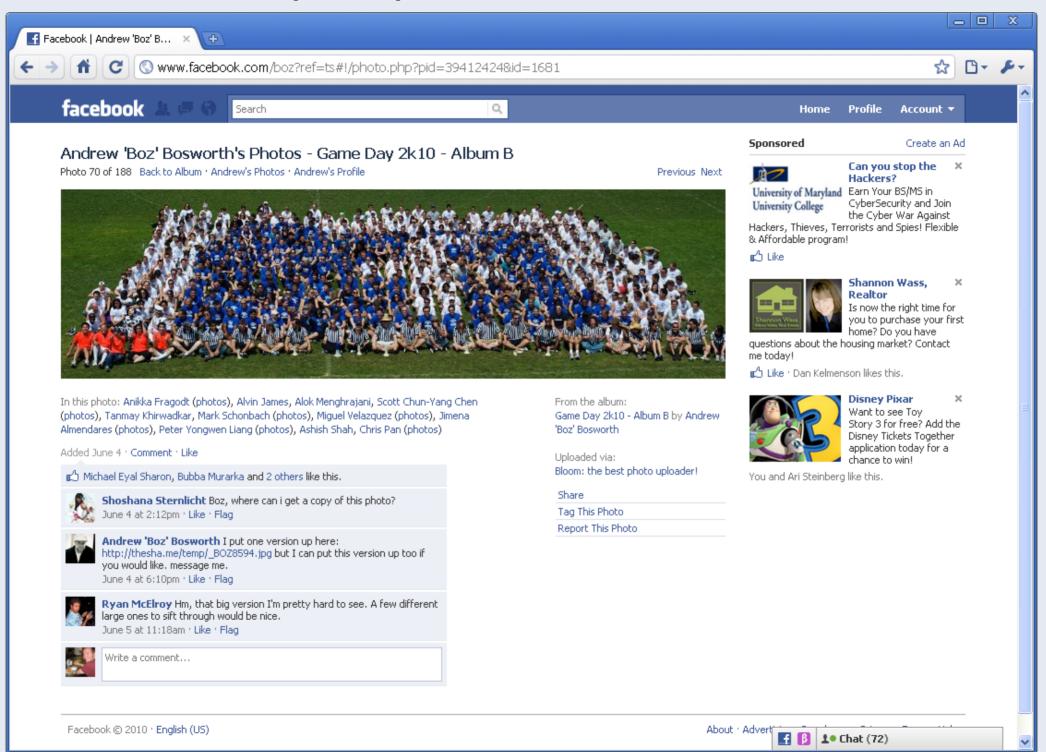
## Time-to-Interact (TTI)



## Time-to-Interact (TTI)



### Time-to-Interact (TTI)



#### Three frontend optimizations at Facebook

Quickling

transparently ajaxifies the whole web site

**PageCache** 

caches user-visited pages in browser

**BigPipe** 

pipelines Facebook website

facebook

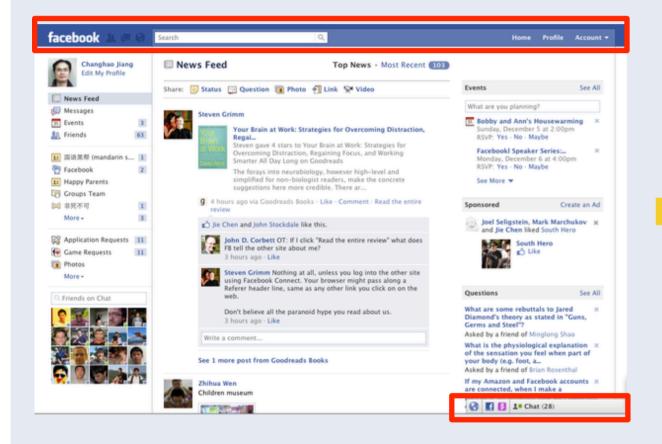
Quickling: Ajaxify the Facebook site

#### Motivation: remove redundant work



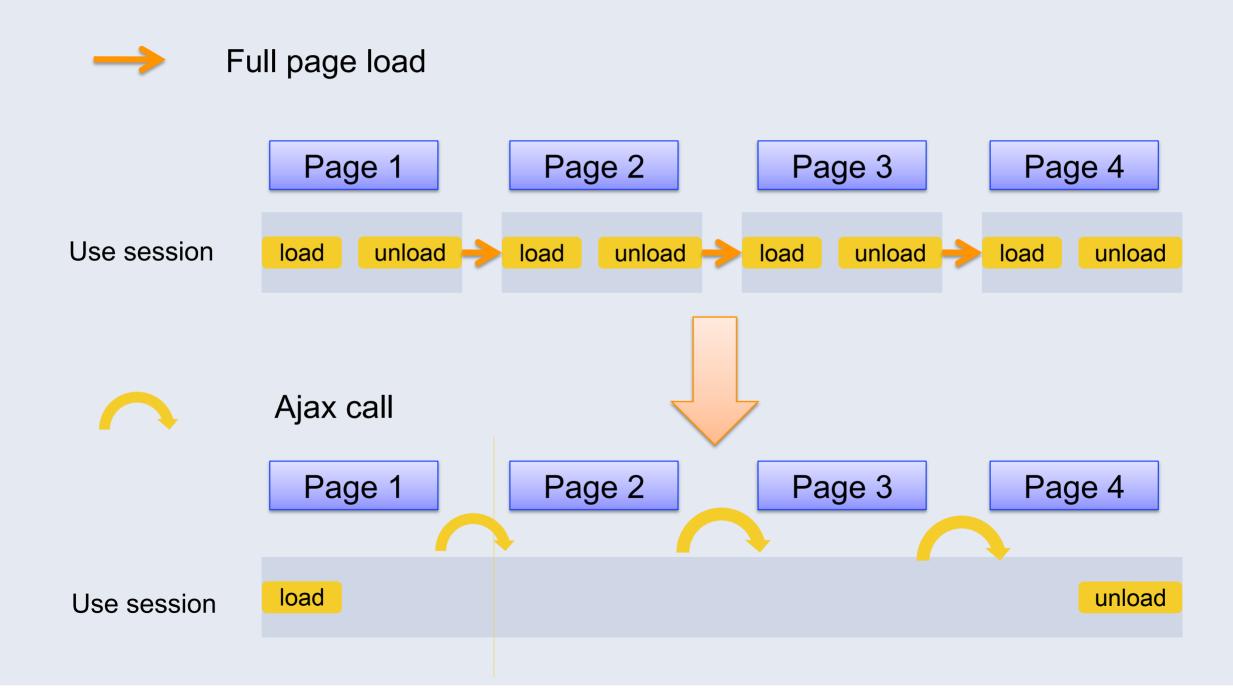


#### Motivation: remove redundant work





#### Remove redundant work via Ajax



### How Quickling works?



- 1. User clicks a link or back/forward button
  - 2. Quickling sends an ajax to server
    - 3. Response arrives

4. Quickling blanks the content area

5. Download javascript/CSS

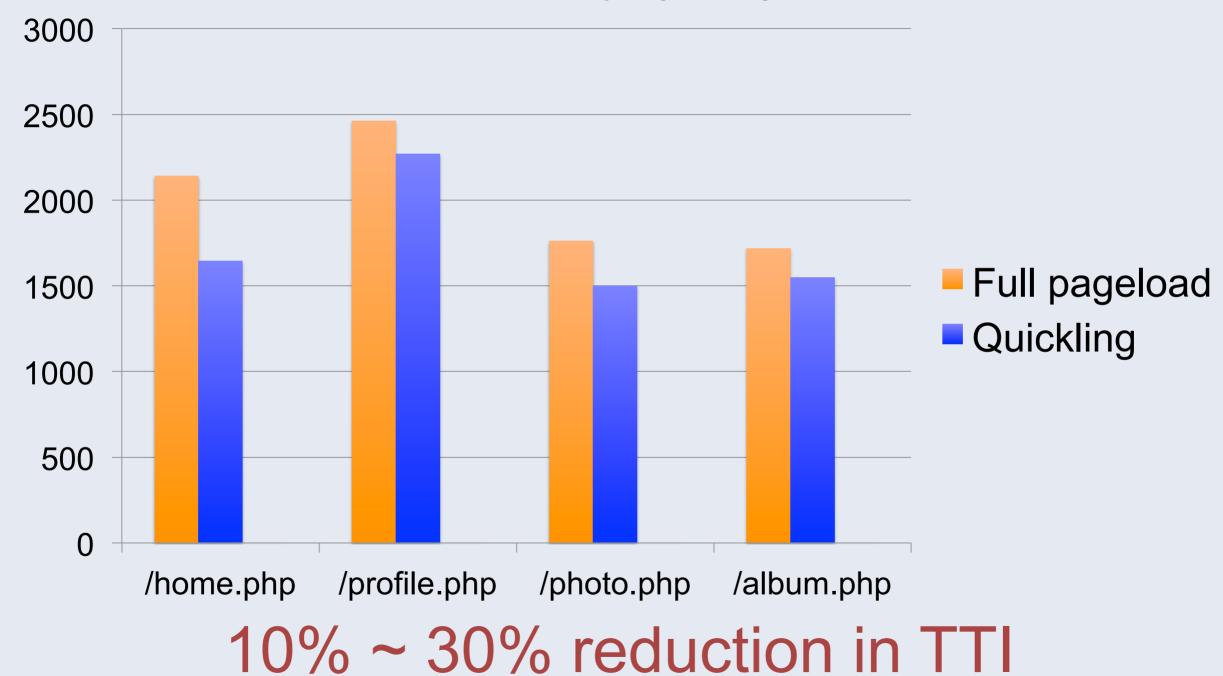
6. Show new content



#### Implementation details

- Link Controller
- HistoryManager
- Bootloader
- Busy Indicator
- CSS Unloading
- Permanent link support
- Resetting timer functions

#### User perceived latency (TTI) reduction

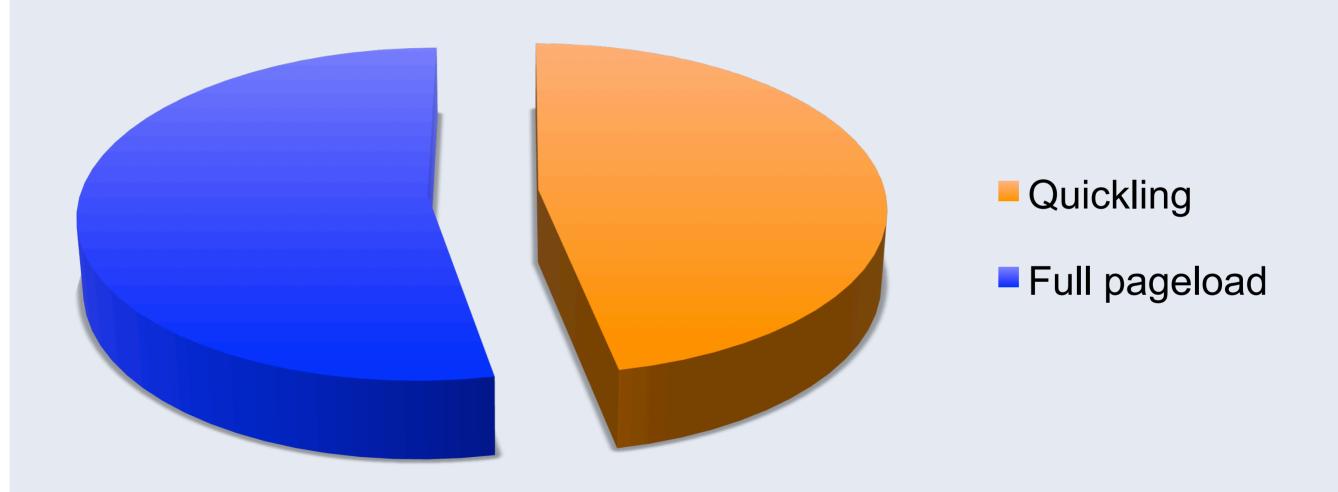


#### Server page generation time reduction



Quickling requests are 20% ~ 30%

## 45% of page hits are Quickling requests



#### Quickling conclusion:

 Reduce user perceived latency by more than 10%

Save 10% of Facebook's data center cost!

facebook

PageCache: Cache visited pages at client side

#### Motivation – temporal locality

- Some pages are likely to be revisited soon
  - E.g. home page (re)visited every 3 page hits:
  - home -> profile -> photo -> home -> notes -> home -> photo -> photo
- Cache them!

#### How PageCache works?



- 1. User clicks a link or back button
  - 2. DinickPiagesientheajakte server
    - 3. Response arrives

3.5 Save response in cache

- 4. Quickling blanks the content area
  - 5. Download javascript/CSS
    - 6. Show new content



#### Challenges

- Real time updates
  - Page content needs to be as fresh as possible
- Cache consistency
  - Cached page should be consistent after state-modifying operations

#### Issue(1): Incremental updates



Cached version



Restored version

#### Issue(1): Incremental updates

#### Poll server for incremental updates via AJAX calls

- Provides an 'onpagecacheRegister' API to developers:
  - Invoked when a page is restored from cache.
  - Used to send AJAX to server for incremental updates, refresh ads, etc.
  - Blank the content area before incremental updates arrive to avoid flash of stale contents.

### Issue(2): In-page writes





Restored version

#### Issue(2): In-page writes

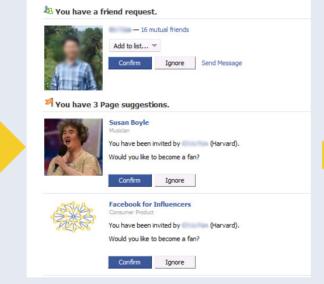
#### Record and replay

- Automatically record all state-changing operations in a cached page
  - All AJAX calls using 'POST' method, or those explicitly specified 'replayable' will be added to cached content.
- Automatically replay those operations when cached page is restored.
  - Cached AJAX handler callback functions already contain all the contextual information about the recorded operations due to JavaScript closure.

## Issue(3): Cross-page writes



Cached version



State-changing op



Restored version

#### Issue(3): Cross-page writes

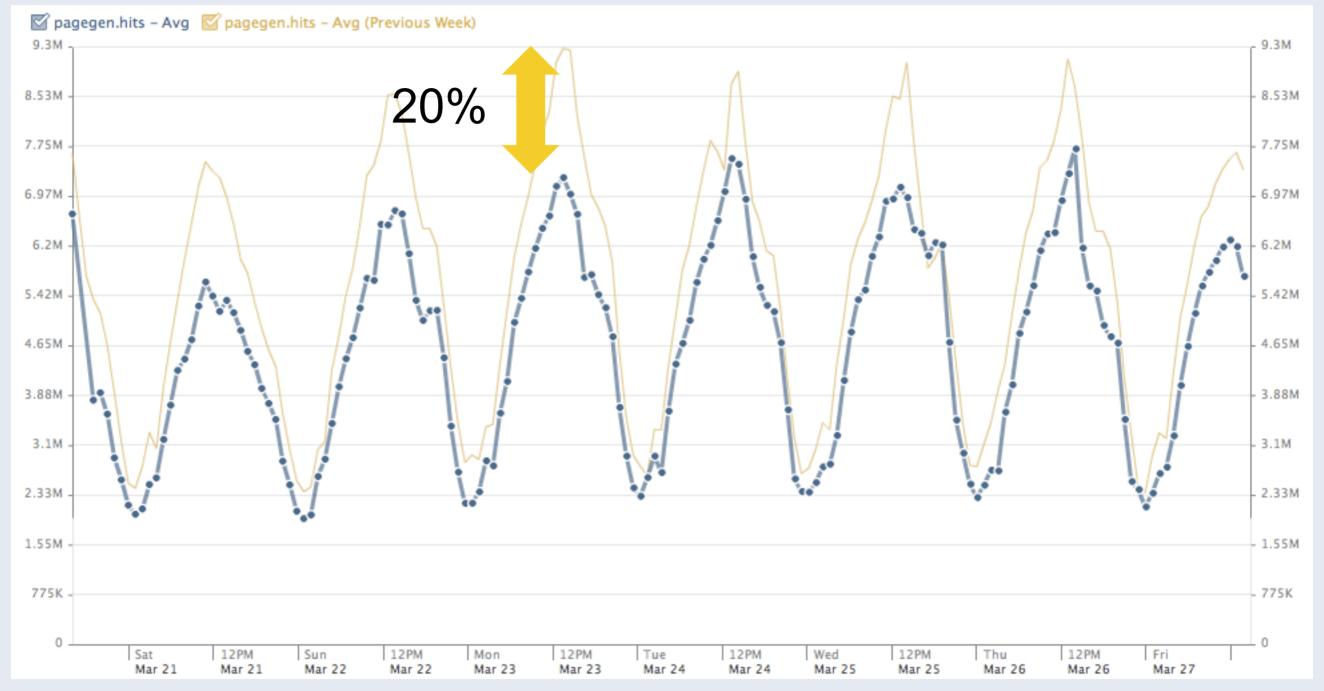
#### Server side invalidation

- Instrument server-side database API, whenever a write operation is detected, send a signal to the client to invalidate the cache.
- The signal (a.k.a. cache invalidation message) contain information about what persistent data has changed.
- Upon receiving cache messages, the client side can flush the whole cache or refresh only those affected page components via AJAX.

### User perceived latency (TTI) redunction



#### Server cost redunction



~20% savings on page hits to home page

#### PageCache conclusion:

- 10X speedup in user perceived speed!
- Save 2% of Facebook's data center cost!

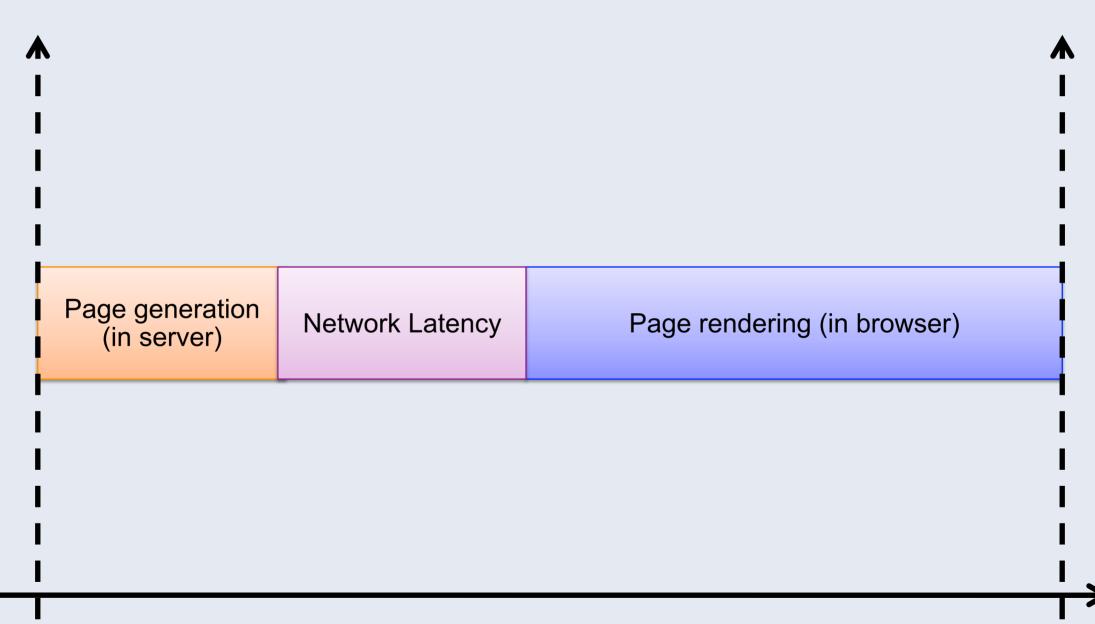
facebook

BigPipe: pipeline Facebook website

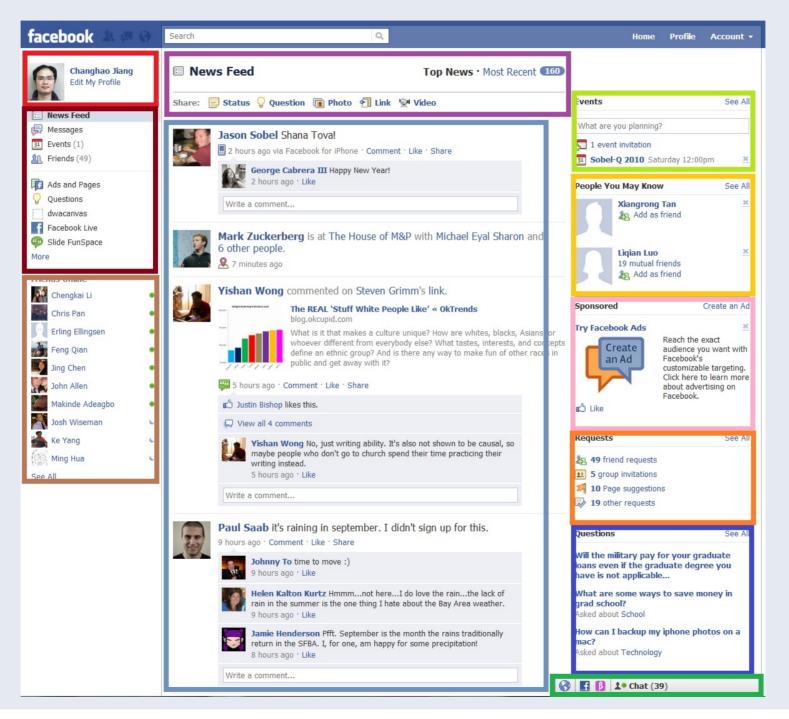
## Facebook page



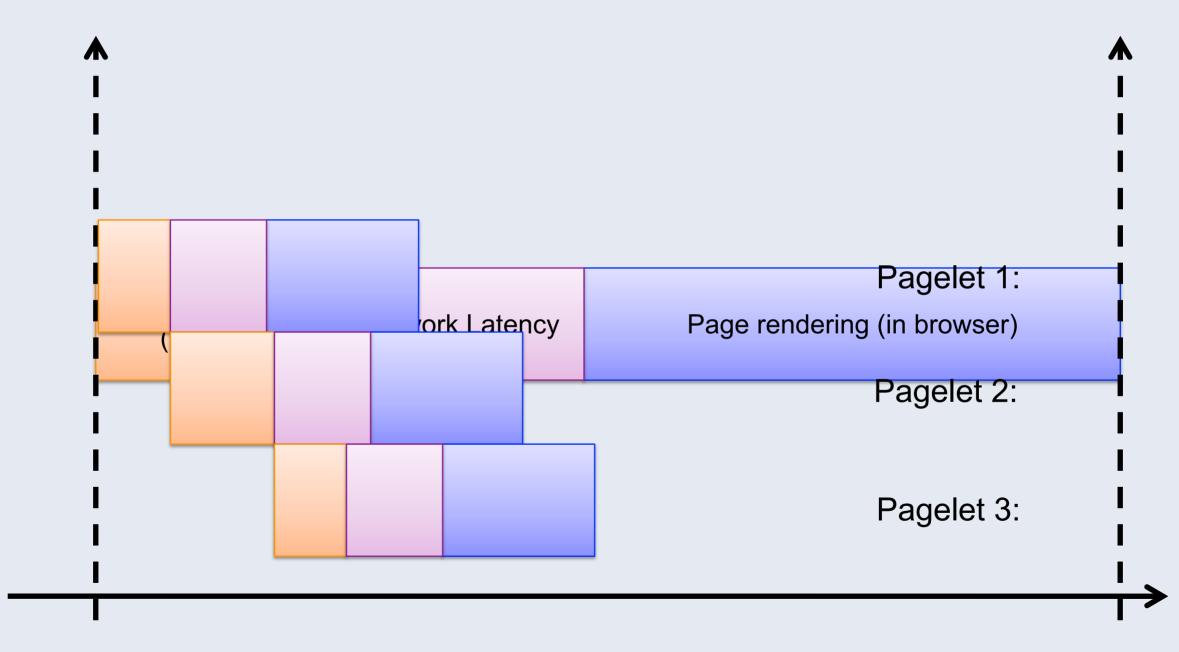
#### **User perceived latency**

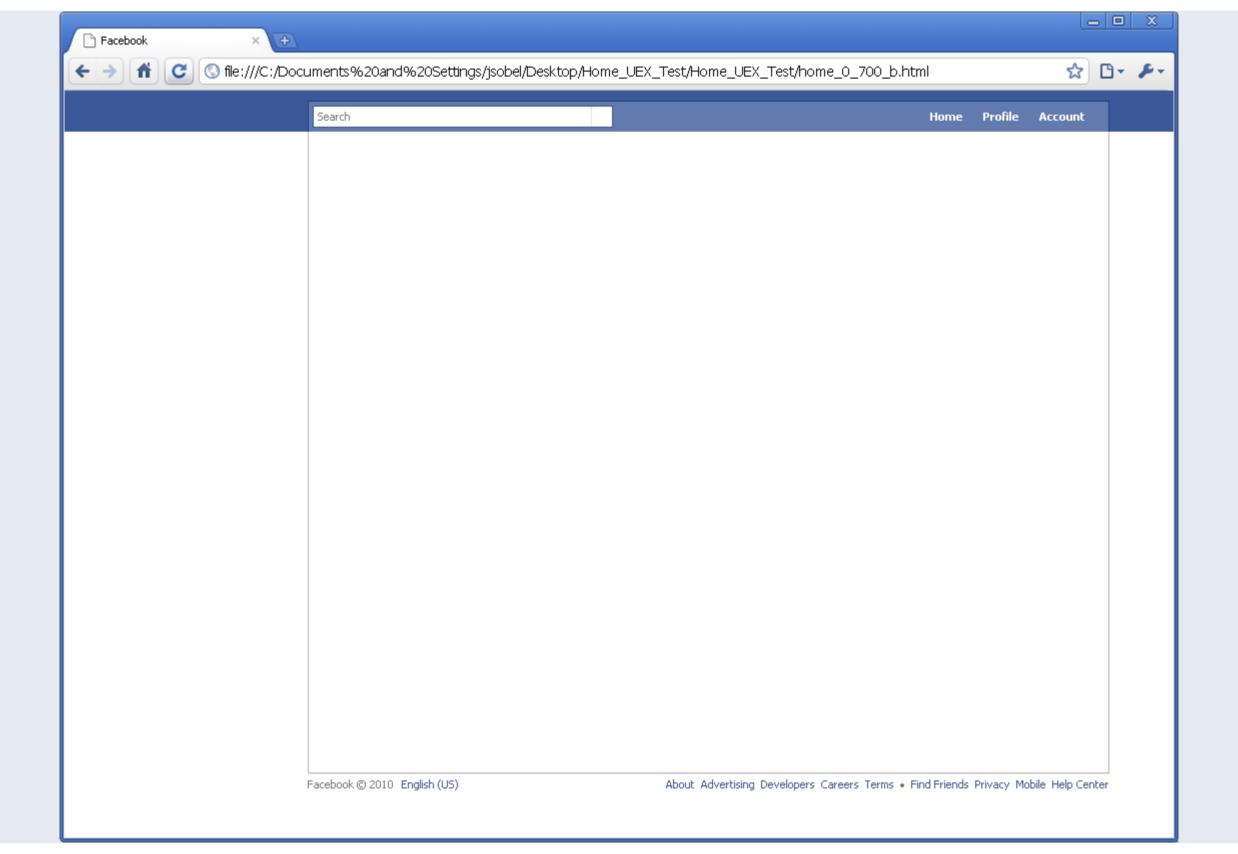


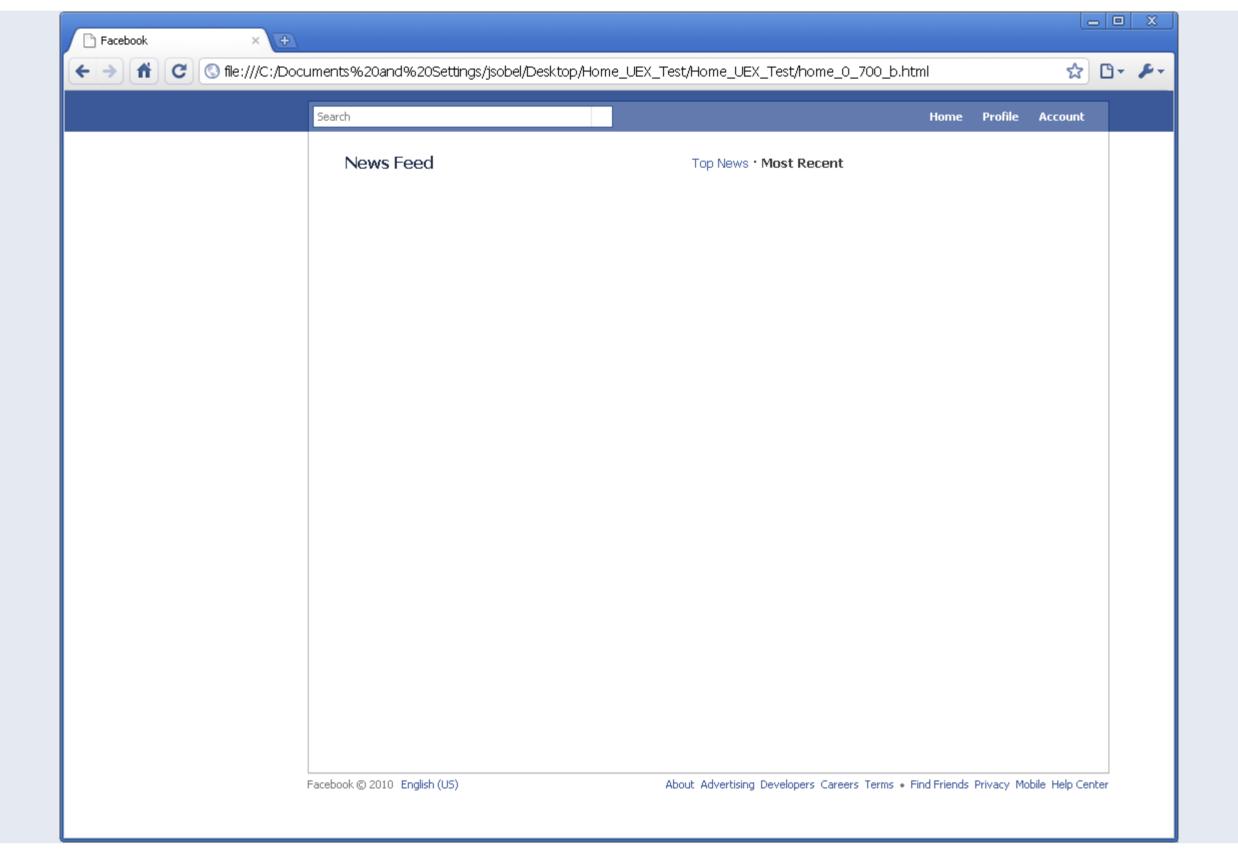
#### **Pagelets**

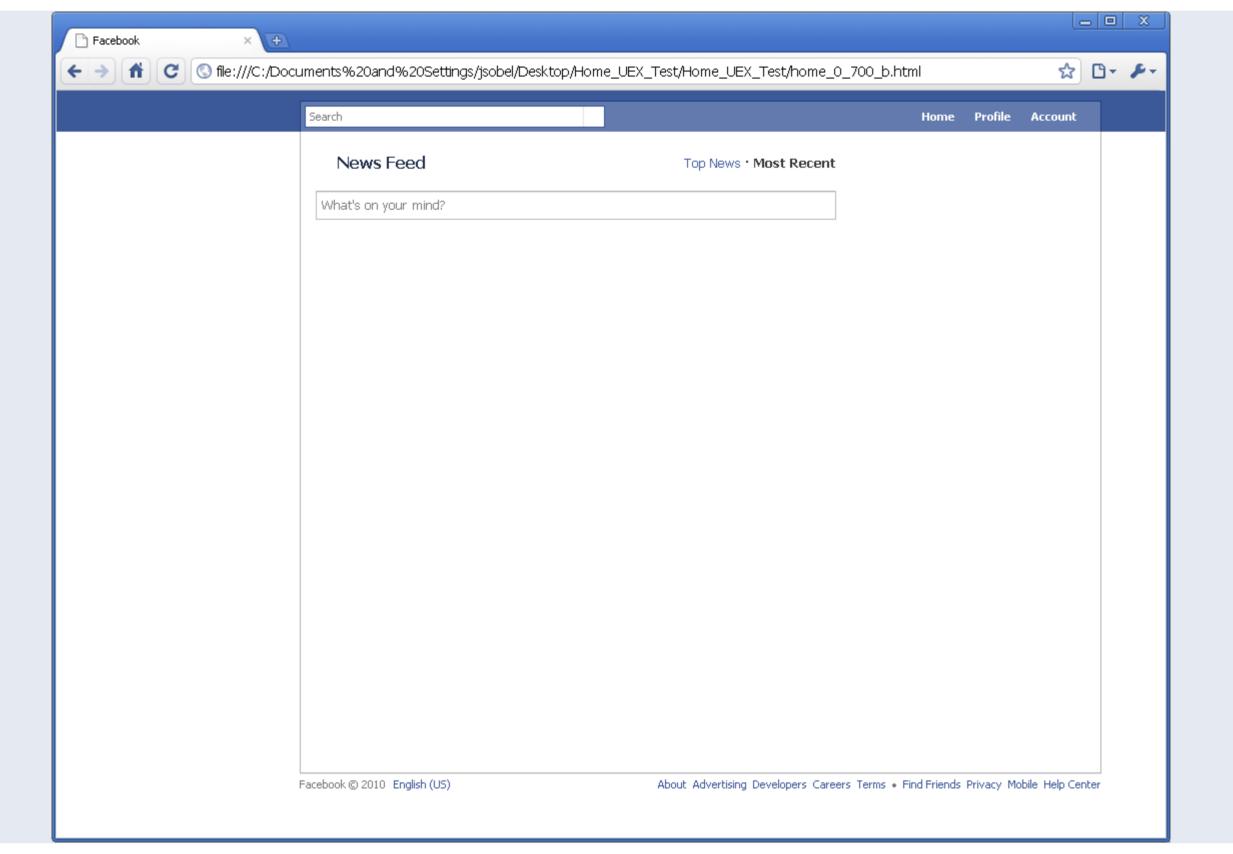


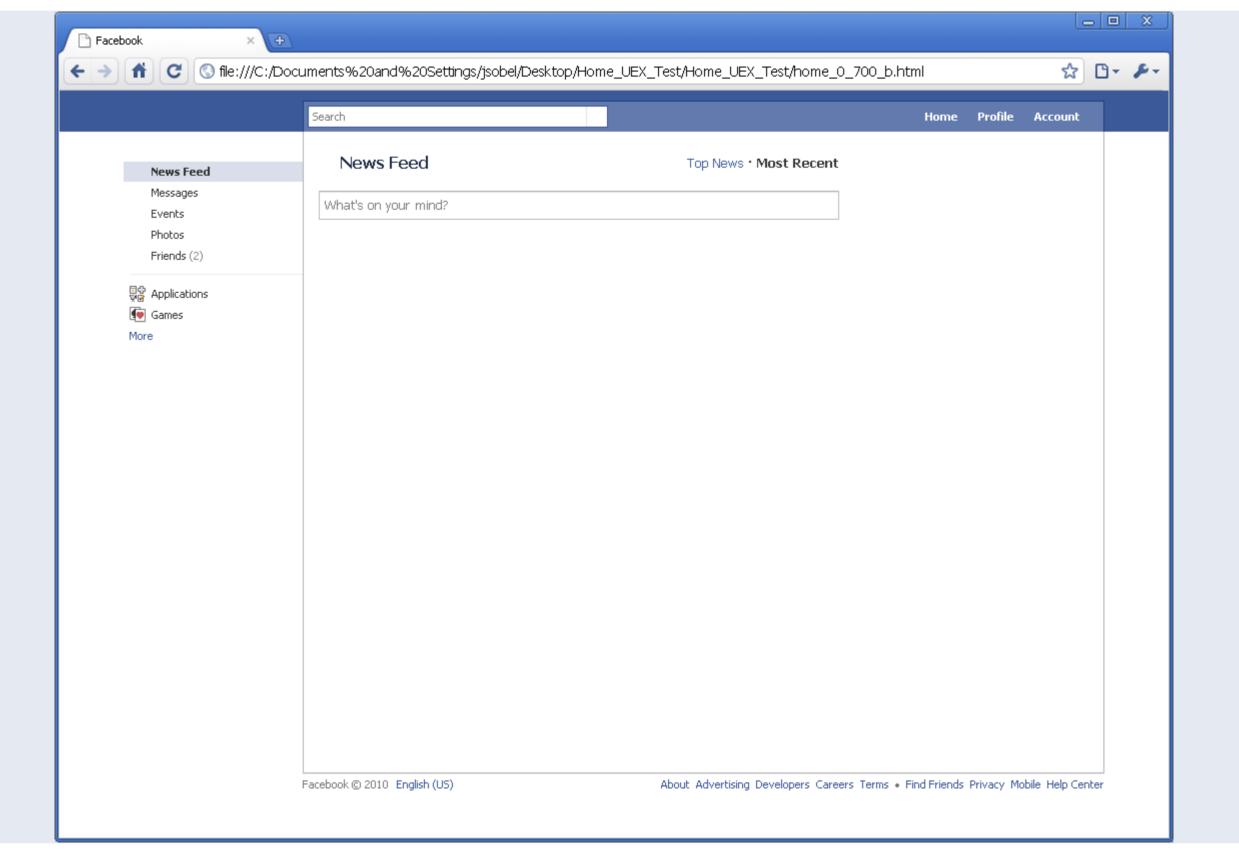
#### **User perceived latency**

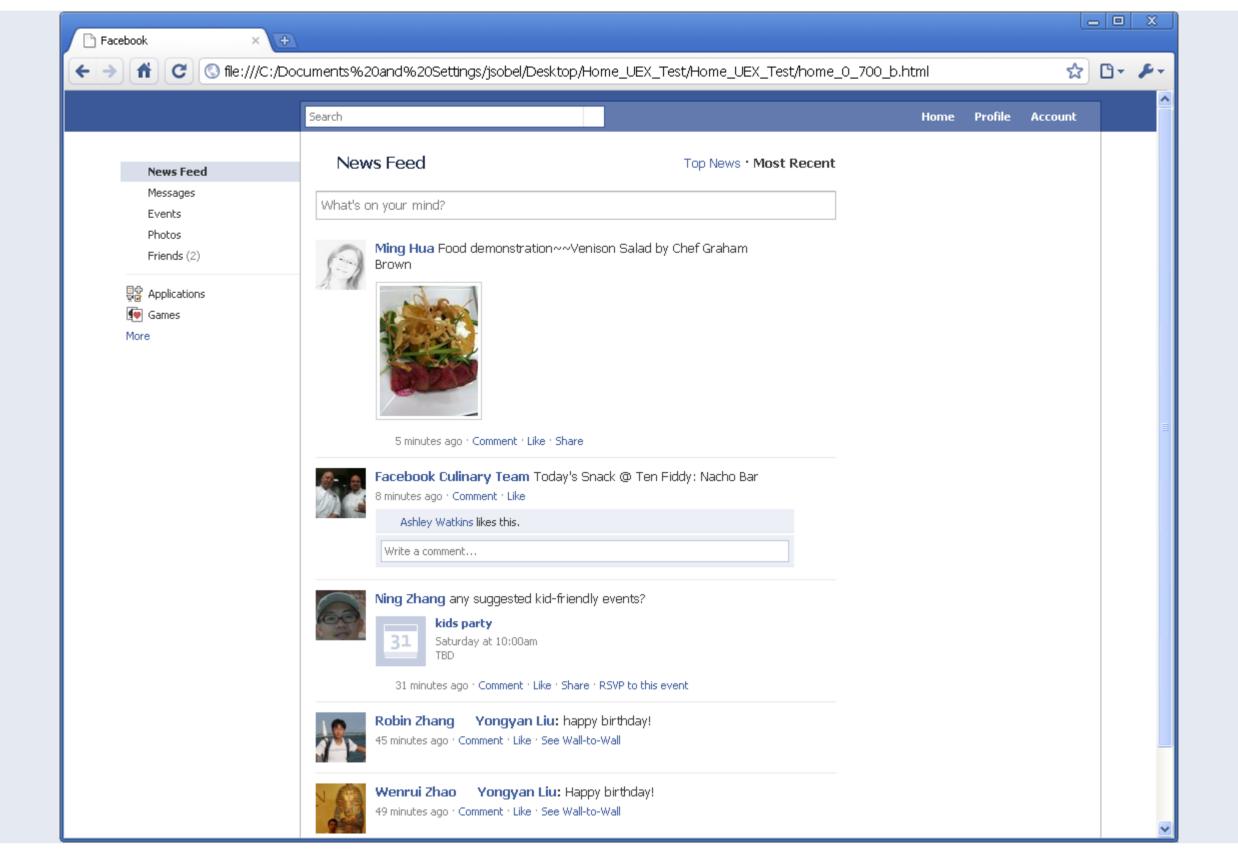


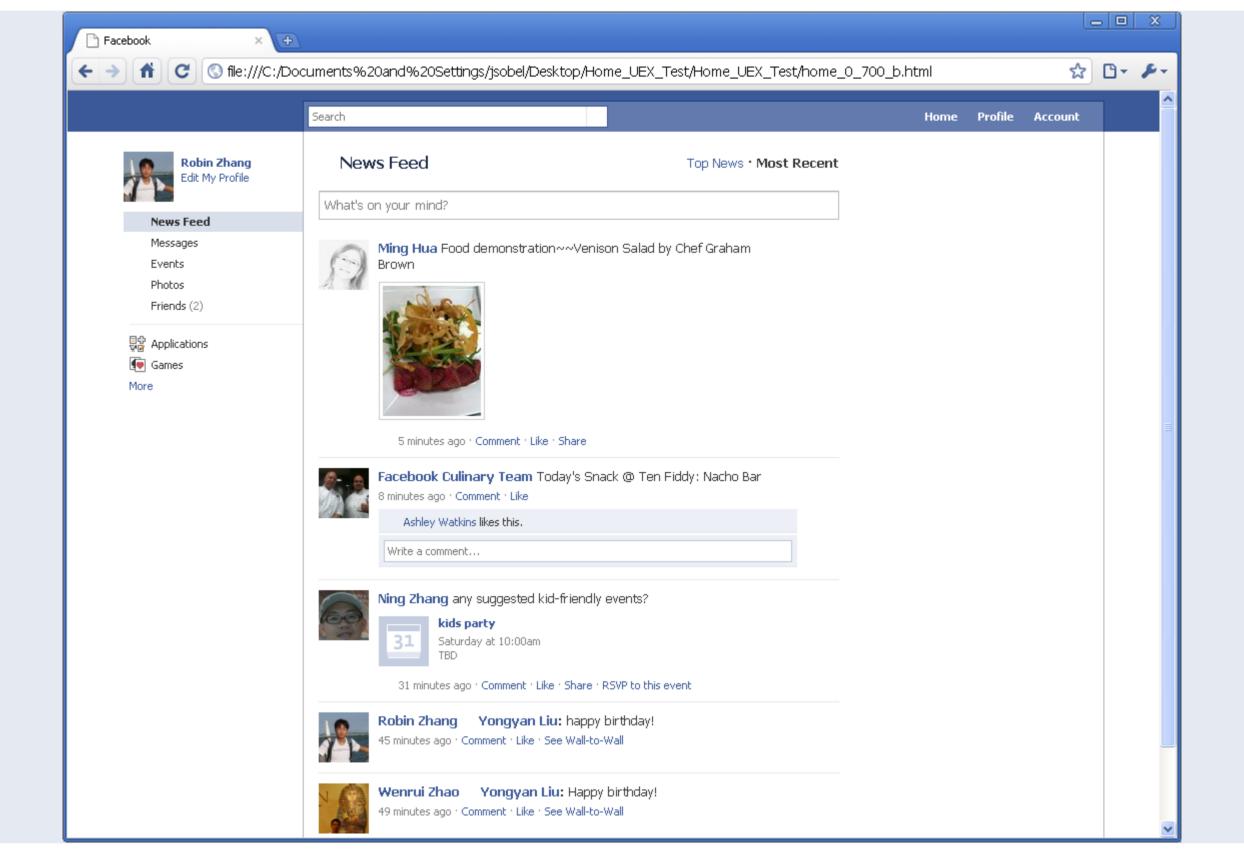


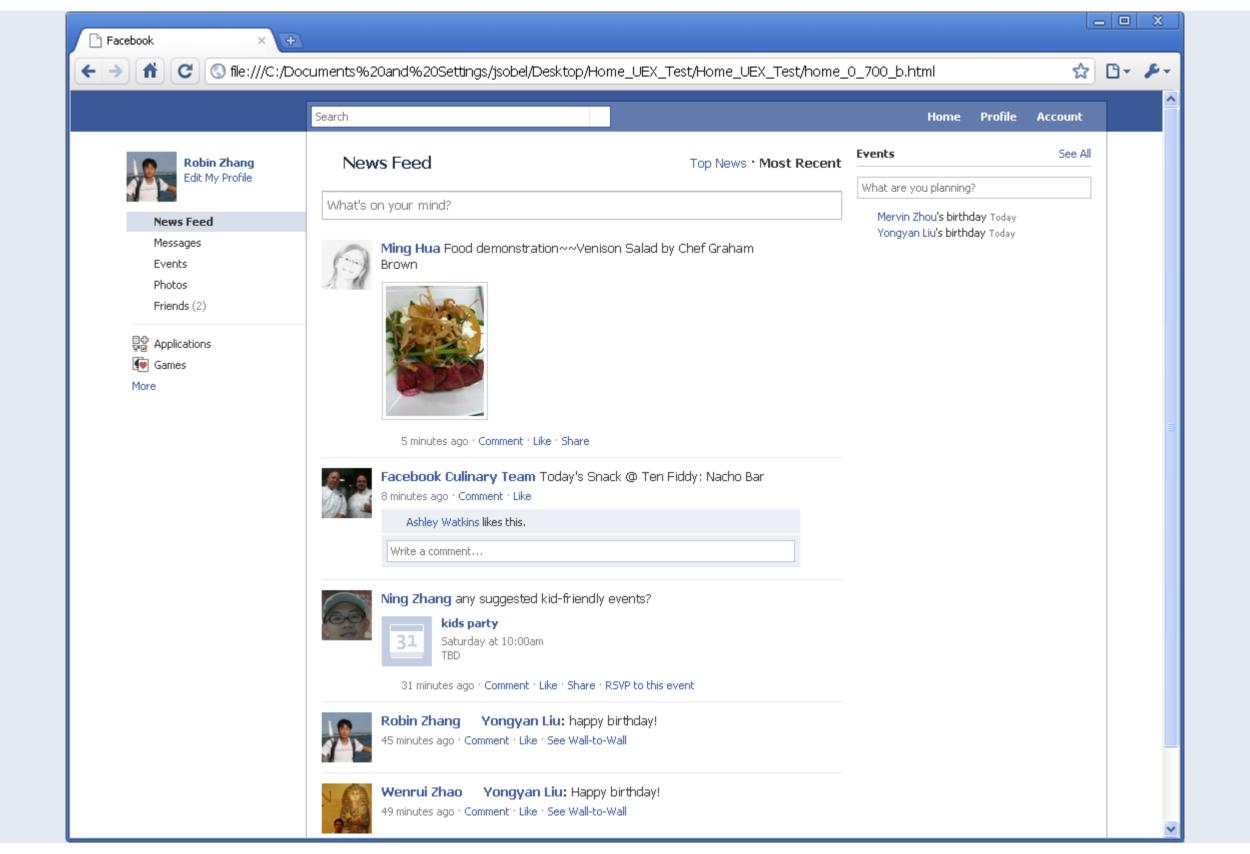


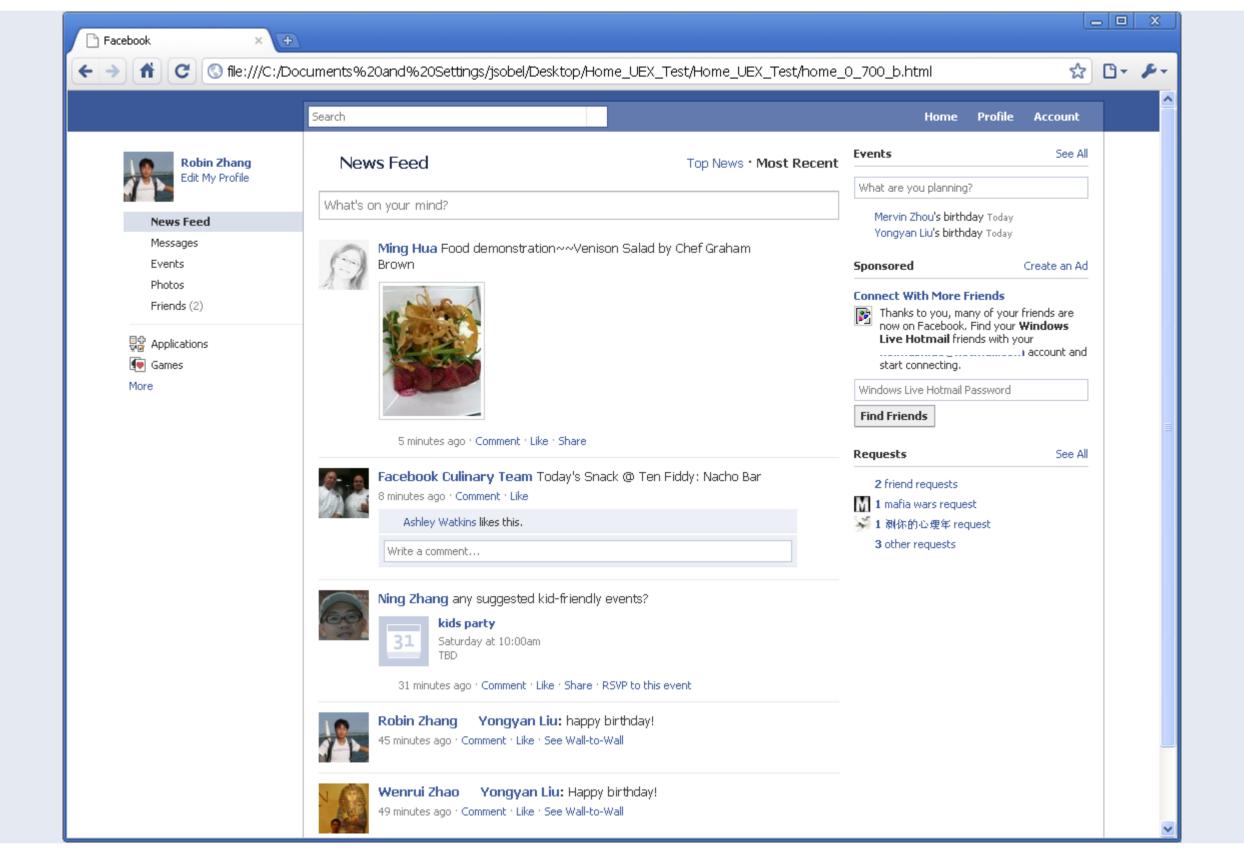


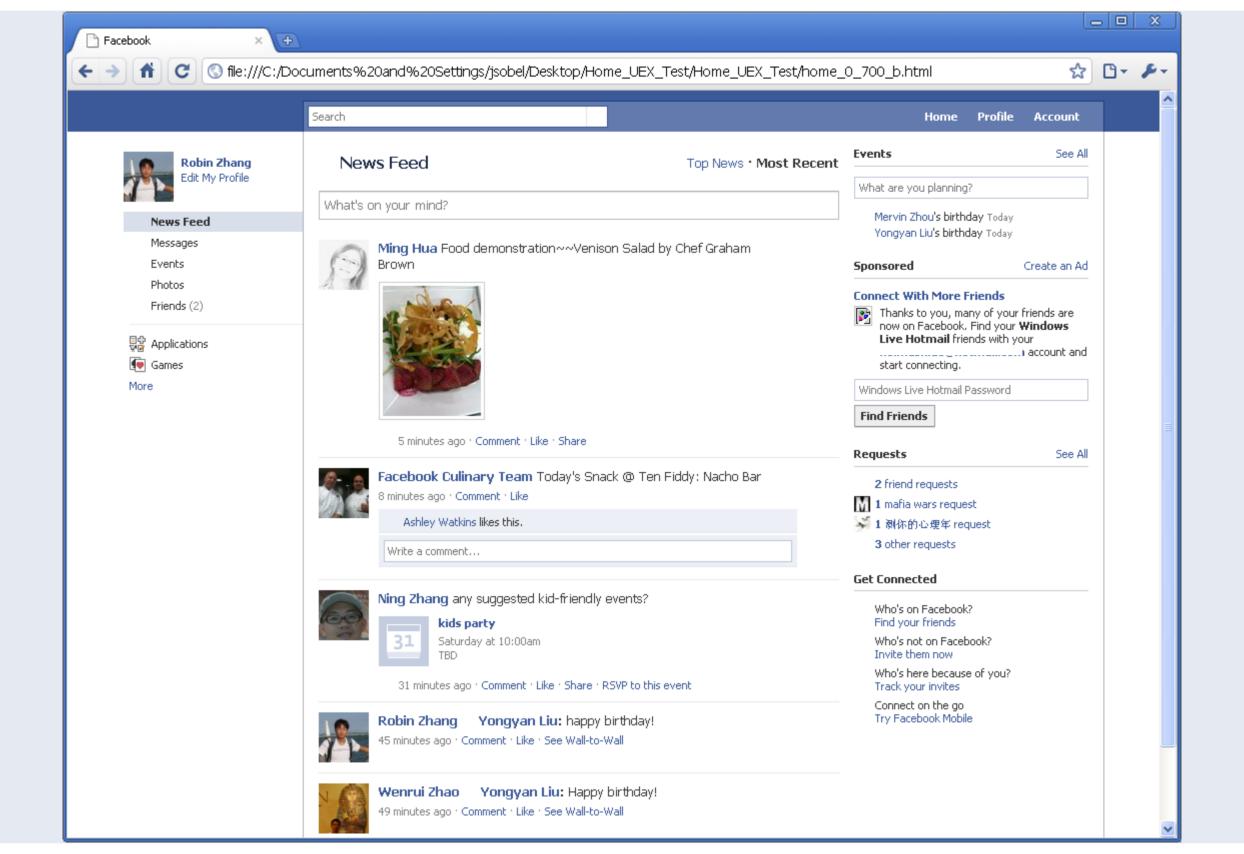


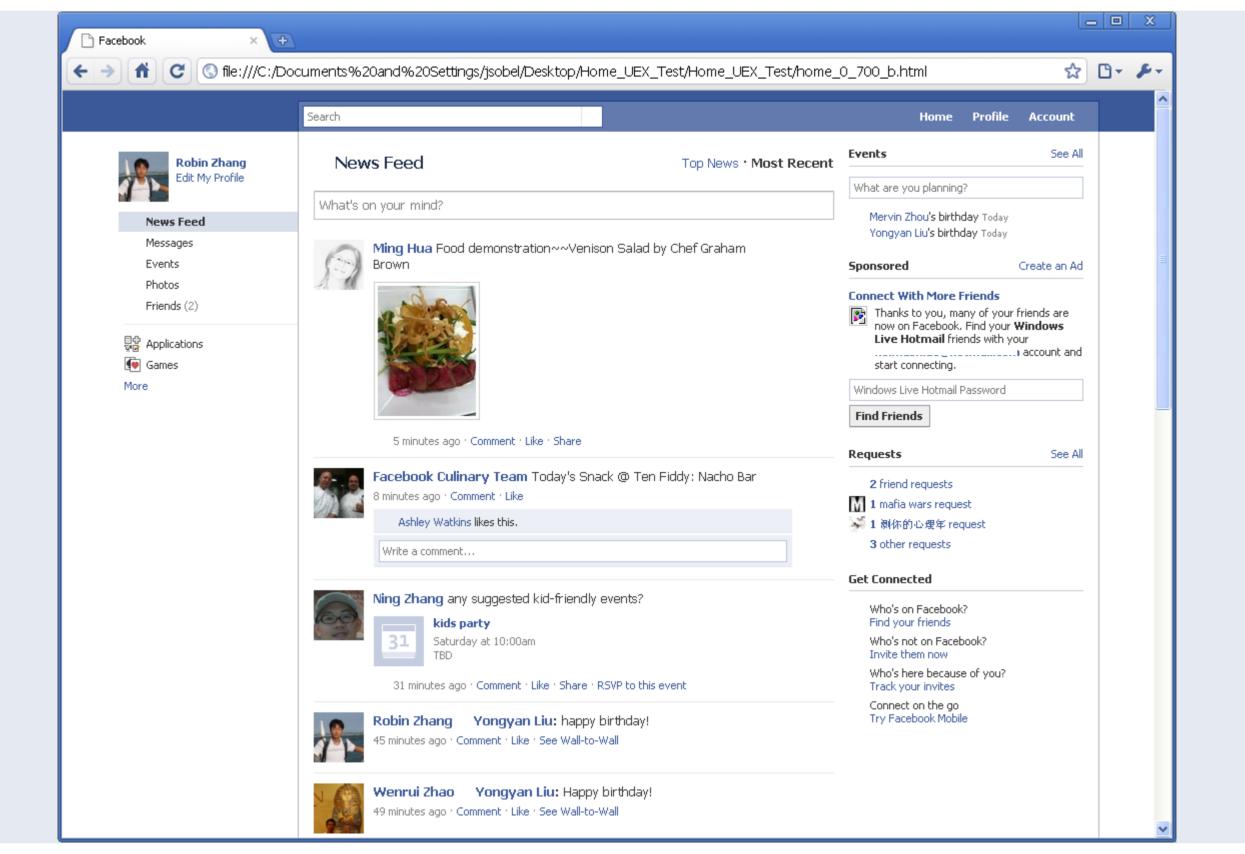


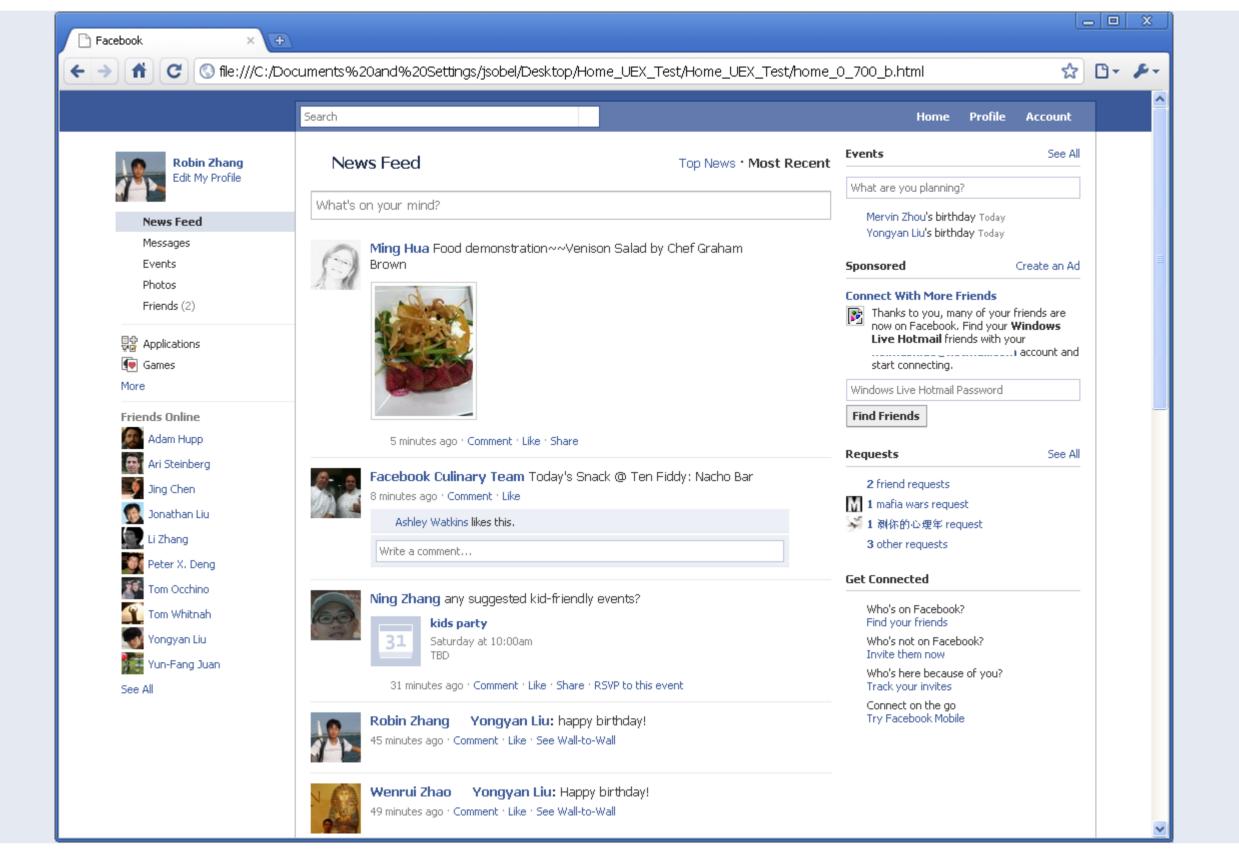


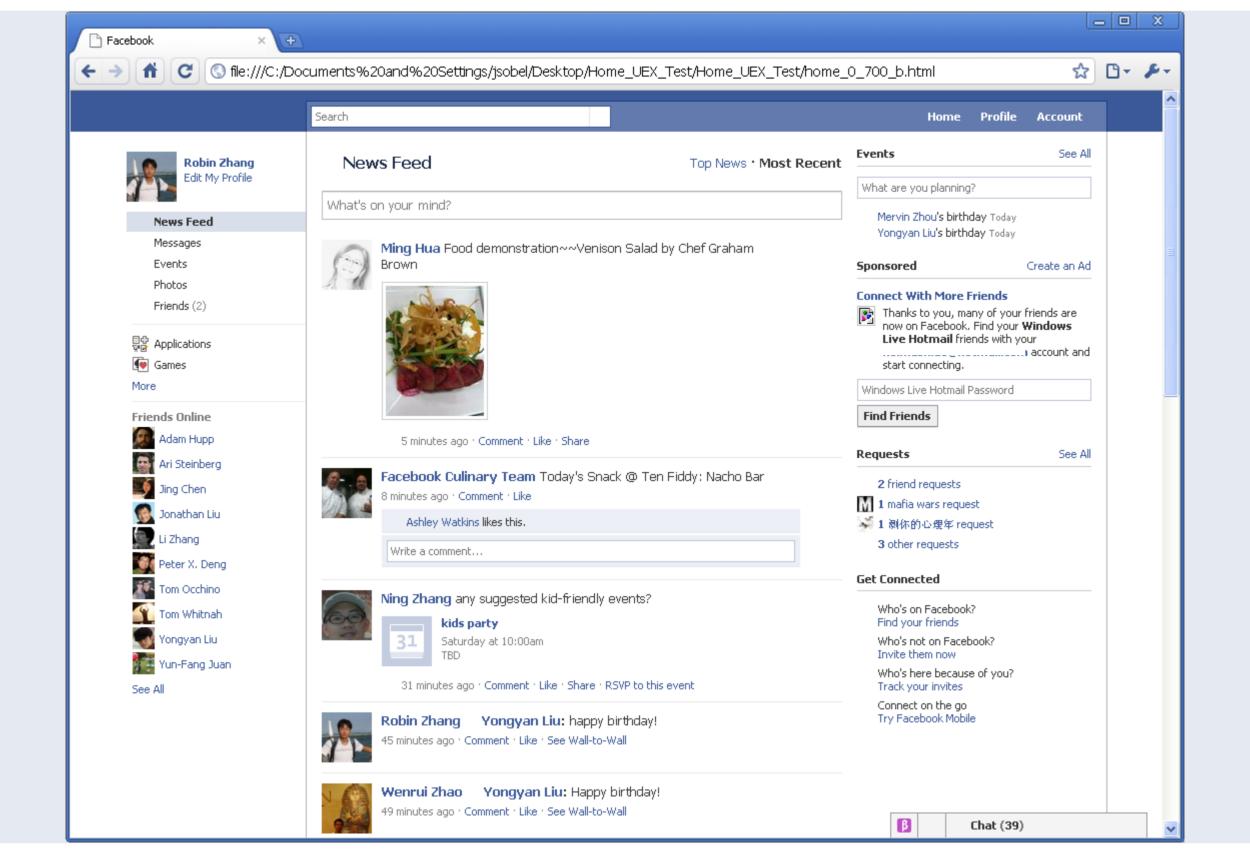




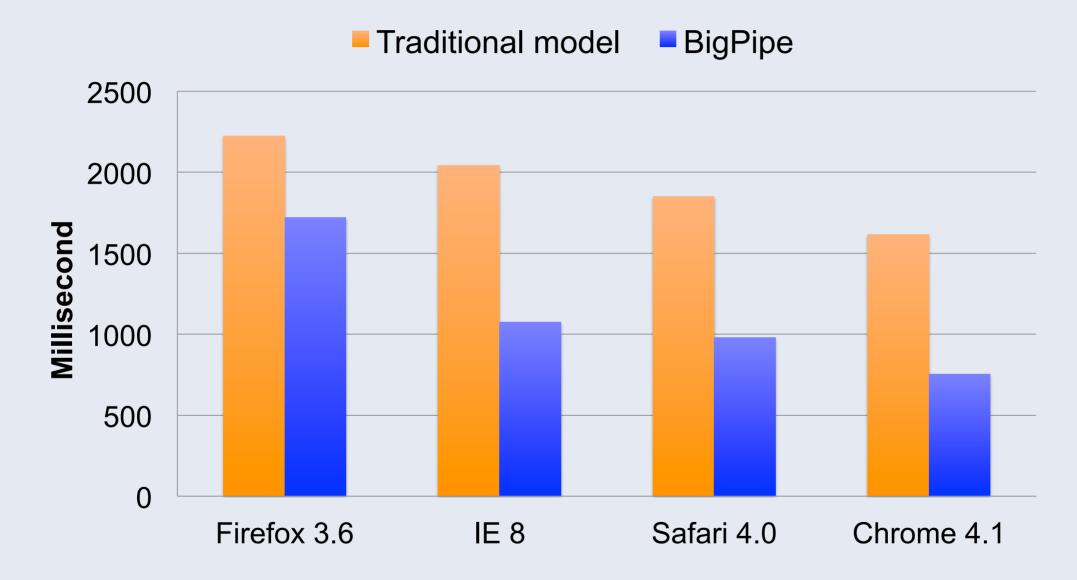








#### TTI improvement



2X improvement in user perceived latency for almost all browsers.

## BigPipe first response

```
head>
<body>
<div id="left column">
</div>
<div id="middle column">
```

```
<html>
<head> <script src="<big pipe js>" />...
<div id="pagelet navigation"></div>
 <div id="pagelet composer></div>
 </div>
```

## Pagelet response 1

```
<script type="text/javascript">
 big pipe.onPageletArrive({
    "id": "pagelet navigation",
   "css": [ < list of css resources > ],
   "is" : [ < list of JavaScript resources > ],
    "content": <html>
    "onload": [JavaScript init code]
});
</script>
```

# Pagelet response 2

```
<script type="text/javascript">
 big pipe.onPageletArrive({
   "id": "pagelet composer",
   "css": [ < list of css resources > ],
   "is" : [ < list of JavaScript resources > ],
   "content": <html>
   "onload": [JavaScript init code]
});
</script>
```

## Pagelet example

```
class AdsPagelet extends PageletResponse
  public function init () {
     // initialize pagelet
  public function prepare() {
     // data fetching
  public render() {
     // generate pagelet response.
```

#### Pagelet programming model

- Self contained
  - HTML, JavaScript, CSS, onloadRegister
- Advanced features:
  - Pagelet hierarchy
  - Phased rendering
  - Cross-pagelet dependencies
    - data dependency
    - display dependency
    - JavaScript dependency

```
// Step 1: create BigPipe instance
$big pipe = BigPipe::getInstance(BigPipeType::PIPELINE);
// Step 2: Specify page layout and pagelet place holders.
$big pipe->setPage(
    '<div id="left_column">
      <div id="pagelet navigation"></div>
     </div>
     <div id="middle column">
       <div id="paglet composer"></div>
       <div id="pagelet stream"></div>
     </div>');
// Step 3: add pagelets to the pipe
$big pipe->addPagelet(
    new UIPagelet()
          ->setSrc('/pagelet/composer.php')
          ->setWrapperId('pagelet composer'));
// Step 4: generate pagelets flush them out.
echo $big pipe->render();
```

```
// Step 1: create BigPipe instance
                     $big pipe = BigPipe::getInstance
                     (BigPipeType::PIPELINE);
                     // Step 2: Specify page layout and pagelet place holders.
                     $big_pipe->setPage(
// Step 1: create BigPipe instance
$big_pipe = BigPipe::getInstance
(BigPipeType::PIPELINE); the pipe
                     $big pipe->addPagelet(
                         new UIPagelet()
                              ->setSrc('/pagelet/composer.php')
                              ->setWrapperId('pagelet composer'));
                     // Step 4: generate pagelets flush them out.
                     echo $big pipe->render();
```

// Step 1: create BigPipe instance

```
// Step 2: Specify page layout and pagelet place
holders.
$big pipe->setPage(
    '<div id="left column">
       <div id="pagelet navigation"></div>
     </div>
     <div id="middle column">
        <div id="paglet composer"></div>
        <div id="pagelet stream"></div>
     </div>
// Step 4: generate pagelets flush them out.
            echo $big pipe->render();
```

// Step 1: create BigPipe instance

```
$big pipe = BigPipe::getInstance(BigPipeType::PIPELINE);
// Step 3: add pagelets to the pipe
$big pipe->addPagelet(
      new UIPagelet()
       ->setSrc('/pagelet/composer.php')
       ->setWrapperId('pagelet_composer'));
           new UIPagelet()
               ->setSrc('/pagelet/composer.php')
               ->setWrapperId('pagelet composer'));
       // Step 4: generate pagelets flush them out.
       echo $big_pipe->render();
```

```
// Step 1: create BigPipe instance
           $big pipe = BigPipe::getInstance(BigPipeType::PIPELINE);
           // Step 2: Specify page layout and pagelet place holders.
           $big pipe->setPage(
              '<div id="left_column">
// Step 4: generate pagelets flush them out.
echo $big pipe->render();
           // Step 3: add pagelets to the pipe
           $big pipe->addPagelet(
               new UIPagelet()
                    ->setSrc('/pagelet/composer.php')
                    ->setWrapperId('pagelet composer'));
           // Step 4: generate pagelets flush them out.
           echo $big pipe->render();
```

#### BigPipe generation modes

- Single flush mode
  - Support search-engine crawlers
  - Support clients without JavaScript
- Pipeline mode
  - Generate and flush pagelets sequentially (default mode)
- Parallel mode
  - Generate pagelets in parallel and flush them out-of-order

- Encapsulation
  - Hide implementation details
- Intuitive
  - Easy to understand mental model for developers
- Flexible
  - Different modes good for different use case.

#### BigPipe advanced features

#### AjaxPipe

- Use iframe to pipeline Ajax response
- Used by Facebook home page's dashboards (photo, groups, etc)

#### WidgetPipe

- Pipeline multiple widgets ("like" button) on third party websites

facebook

Thank you!