Google
Developers

SPDY
It's Here!

Roberto Peon
Software Engineer, Google

Overview

- The Past
- What does the world look like today?
- What is SPDY?
- How to optimize for SPDY
- The future of SPDY
- Stuff using or supporting SPDY
- An example: adding server push

The Past

Near the beginning:

The speed of light was: **C**

(Oops, lets try again)

Near the beginning (of the web):

There was HTTP "0.9"

And then (1996) there was:

HTTP/1.0

Shortly thereafter (1997-1999) there was:

HTTP/1.1

What did a page look like in 1999?   It was:

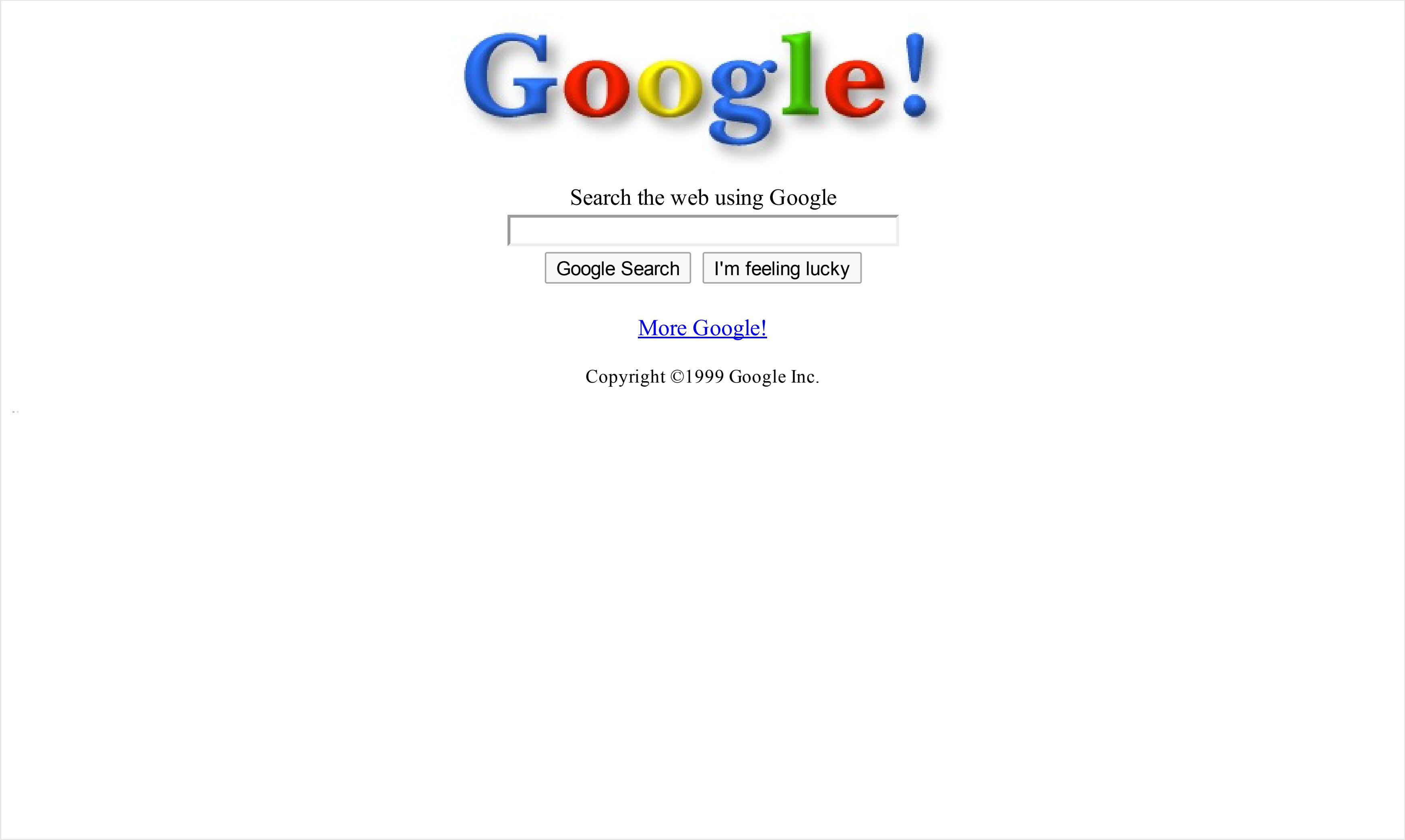
Tiny!

Specifically, around 60k on average

And, at the time, that was probably good enough

source: www.pantos.org

Google in 1999:

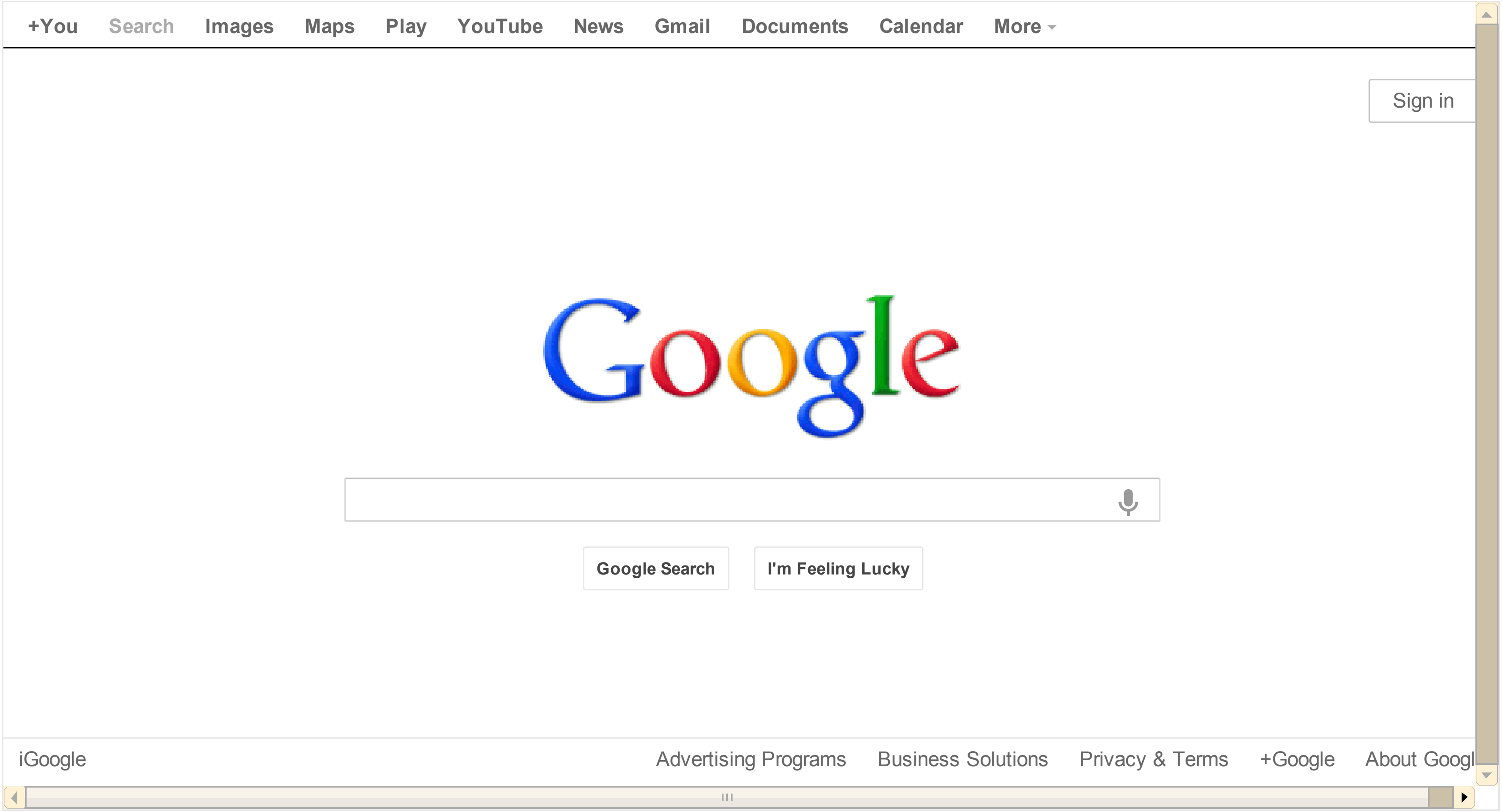


Search the web using Google

Copyright ©1999 Google Inc.

Today

Google in 2012:



Today:

Pages
are
much
much
larger

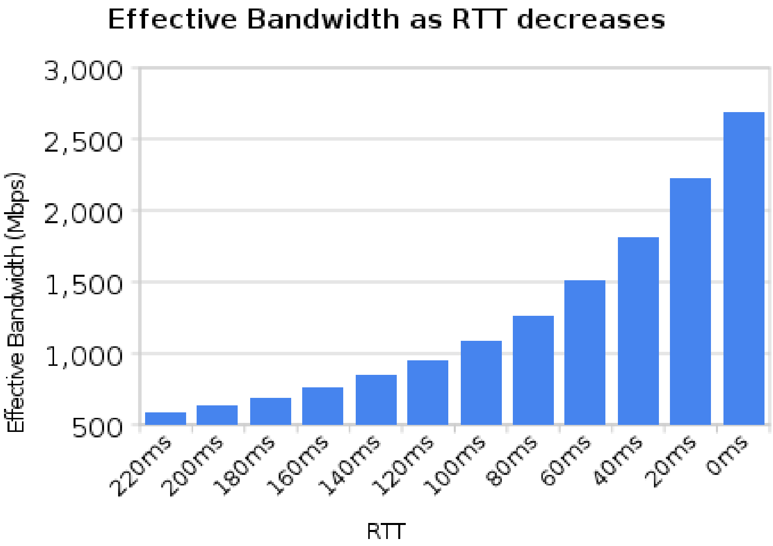
Today:

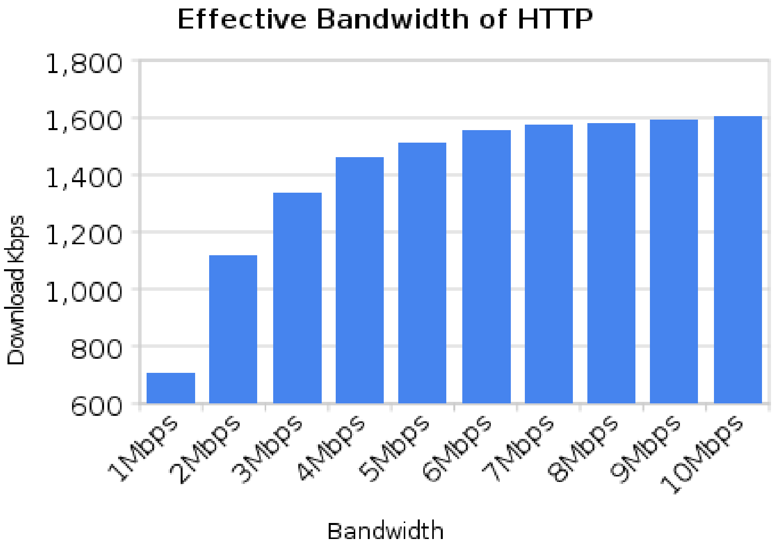
- Pages are composed of more resources
- They use a bunch of domains
- They're far more dynamic
- Security is a bigger concern

Today:

The speed of light is **still: C**

And that really matters... because speed-of-light bounds RTT!





2010 vs 2012

	Avg. Page Size	Requests/Page	Domains
Nov 15 2010	702k	74	10
May 15 2012	1059k	84	12

The trend seems to continue towards larger pages using more resources.

source: httparchive.org

But Now...

The stakes are higher...

The internet has become more important

More people depend on it

It's big money!

SPDY: It's here!

SPDY is a session-layer replacement for HTTP's connection handling

So.... Why?

The idea is that SPDY can help make sites:

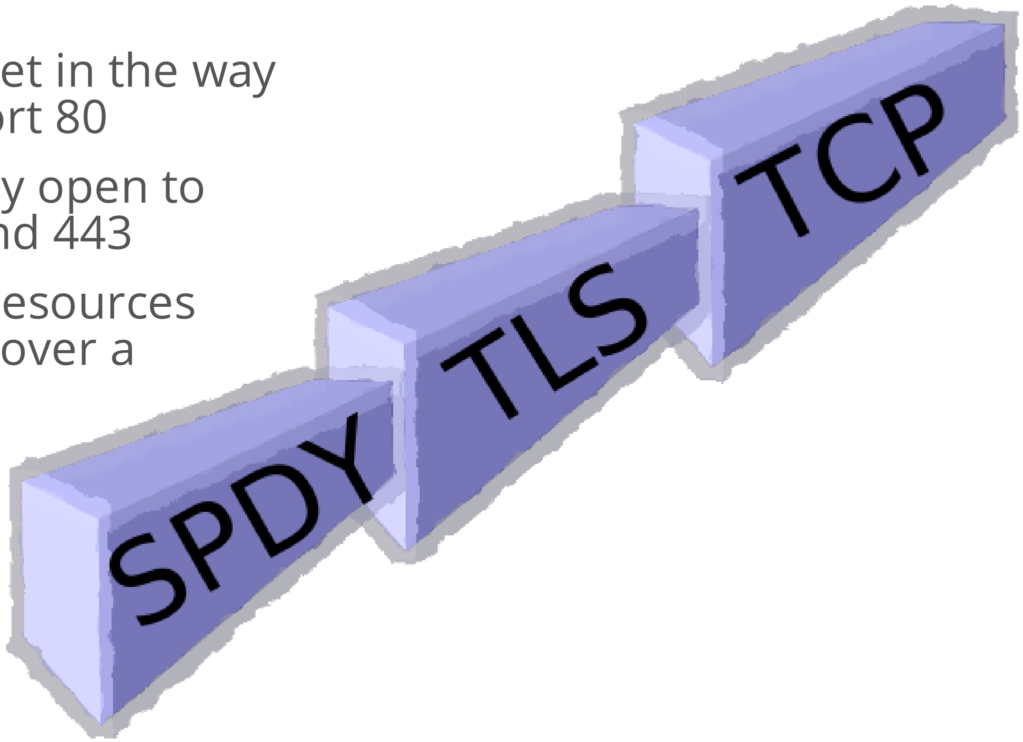
- Faster
- More secure
- And, long-term, it can help to do it with less effort

- (but no guarantees...)

How does it work?

SPDY:

- Is always over TLS
  - Point-to-point privacy by default
  - Transparent proxies get in the way of anything else on port 80
  - Firewalls aren't reliably open to ports other than 80 and 443
  - Allows for loading of resources with an HTTP scheme over a TLS connection



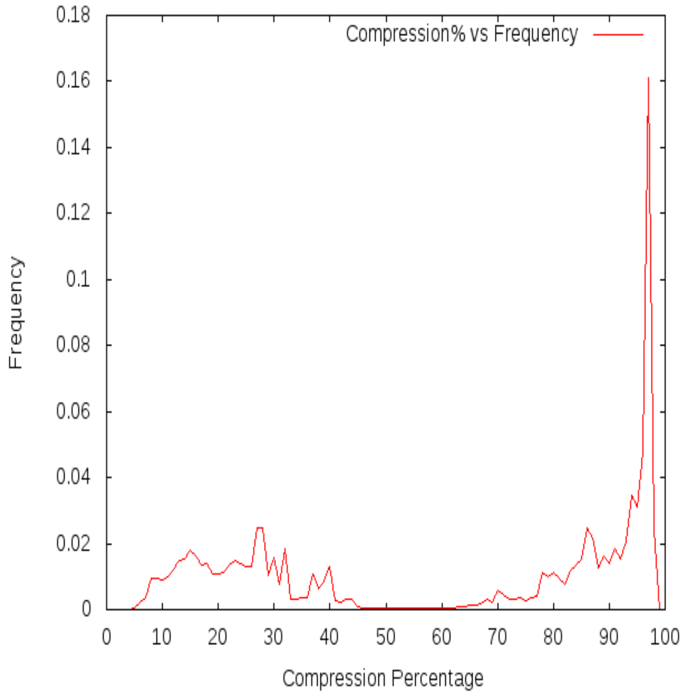
SPDY:

- Is always over TLS
- Does Headers Compression
  - Much of the headers are repeated.
  - Much of the headers are repeated.
  - User-Agent
  - Cookie
  - HTTP methods
  - HTTP version string
  - All of those "\r\n"s
  - etc.

Imagery supplied by PhotoObjects.net/Getty Images

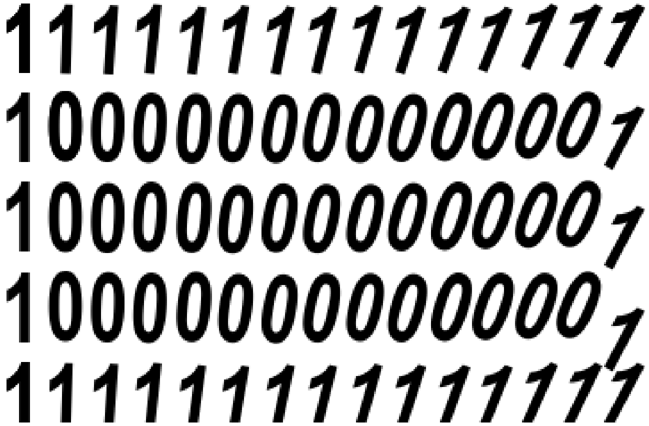
SPDY:

- Is always over TLS
- Does Headers Compression
  - Reduces bandwidth required for headers
  - 10-35% size reduction typical for the first request
  - 80-97% size reduction on longer-lived connections
  - Especially useful in the client to server direction since upload bandwidth is often smaller
  - Think: mobile devices...



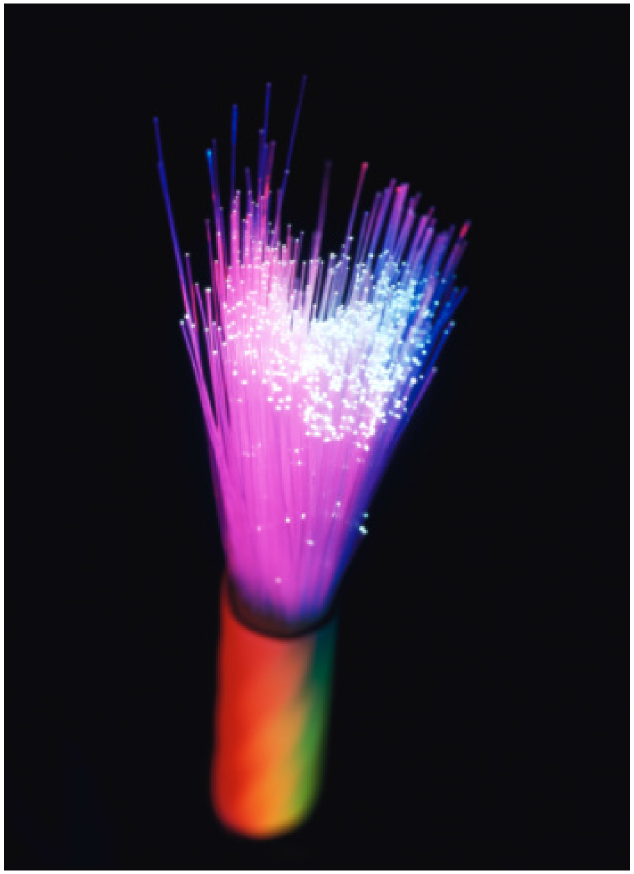
SPDY:

- Is always over TLS
- Does Headers Compression
- Is Binary Framed
    - It is faster to parse
    - Less error prone,
      Easier to implement right!



SPDY:

- Is always over TLS
- Does Headers Compression
- Is Binary Framed
- Is Multiplexed
  - Many (1,073,741,824) independent streams per TCP connection
  - Either side can create a new stream at any time



Imagery supplied by Stockbyte/Getty Images

SPDY:

- Is always over TLS
- Does Headers Compression
- Is Binary Framed
- Is Multiplexed
- Has Full-Duplex Interleaved and Prioritized Streams
  - Prioritization allows the browser to forget about request heuristics and just send all the requests
  - Interleaving allows "simultaneous" delivery of data on multiple streams
  - Interleaving also allows for high priority data to temporarily 'interrupt' low-priority data

SPDY:

- Is always over TLS
- Does Headers Compression
- Is Binary Framed
- Is Multiplexed
- Has Full-Duplex Interleaved and Prioritized Streams
- Allows for Server Push
  - Unlike data-URLs, it allows resource caching
  - Inlining is annoying
  - Shaves off an RTT as compared to just putting in a link and waiting
  - Making its way into implementations now

Imagery supplied by Comstock/Getty Images

SPDY:

- Is always over TLS
- Does Headers Compression
- Is Binary Framed
- Is Multiplexed
- Has Full-Duplex Interleaved and Prioritized Streams
- Allows for Server Push
  - Uses the same or less bandwidth as inlining
  - Streams can be aborted if the browser realizes it already has the resource

Imagery supplied by Comstock/Getty Images

SPDY:

- Is always over TLS
- Does Headers Compression
- Is Binary Framed
- Is Multiplexed
- Has Full-Duplex Interleaved and Prioritized Streams
- Allows for Server Push
- Uses One TCP Connection
    - Reduces buffer bloat
    - Uses fewer server resources
    - Allows caching of the CWND on the client
    - Allows servers to better prioritize your important traffic

Imagery supplied by Stockbyte/Getty Images

SPDY:

- Is always over TLS
- Does Headers Compression
- Is Binary Framed
- Is Multiplexed
- Has Full-Duplex Interleaved and Prioritized Streams
- Allows for Server Push
- Uses One TCP Connection
- Uses Fewer Packets
    - Thanks to compression and using only one TCP connection

Imagery supplied by Stockbyte/Getty Images

SPDY:

- Is always over TLS
- Does Headers Compression
- Is Binary Framed
- Is Multiplexed
- Has Full-Duplex Interleaved and Prioritized Streams
- Allows for Server Push
- Uses One TCP Connection
- Uses Fewer Packets
- Doesn't Require Rewriting Your Site
    - Many parts of SPDY can help for most sites
    - Some features, like 'server push' take more effort

Imagery supplied by Stockbyte/Getty Images

SPDY Isn't Magic

(the no guarantees bit)

Things that make SPDY less effective:

- Lots of 3rd-party domains with resources you need to render your page
  When you have lots of domains that don't match the original cert, SPDY can't reuse the connection.
  This matters most when comparing its speed to HTTP

- Very few resources without connection reuse
  SPDY may not help you if you have very few resources (<6) and your users navigate away after seeing the page instead of nagivating further in
  Basically, the more "cold" page-loads, the less benefit SPDY gives

- High packet-loss links
  SPDY does worse than HTTP (for today) on links with high RTT and high packet loss.

SPDY vs HTTP?

As compared to SPDY, HTTP has some significant limitations.

- Not secure by default
- No header compression
- No multiplexing
- Not full-duplex
- No prioritization; Browsers must employ "fun" heuristics instead.
- No server push
- No interleaving
- May require more DNS lookups
- Uses far more connections
- Helps cause buffer-bloat

But, Aren't Browsers Getting Smarter?

Yup. They are.

If your user has visited the site before, a smart browser like Chrome may help.

In particular, a browser could remember that visiting domain A means that it will likely use:

6 connections to domain A, 6 connections to domain B, 3 connections to domain C, etc.

For this to work, however, it requires the site to have changed little, for the user to have already visited the site, for the browser to learn these things when the user visited the site, and it still depends on the timely delivery of the DNS resolutions for the other domains.

If the DNS data had a sufficiently high TTL, you don't have to redo the resolutions... but there are good reasons to keep TTLs low such as fast reaction outages or for load balancing.

You probably just saw "Wall 'O Text", right?

The one-sentence breakdown

Yes, they'e getting smarter, but it will likely still be worse that what you can do with SPDY.

The Cost of HTTP Connections

Each connection:

- Requires a new DNS resolution (when going to a new domain)
    - For many people this can be 100+ ms!
- Requires a new 3-way handshake
    - +1 RTT
- If using HTTPS
    - generally +2 RTT, sometimes more
- Connections aren't free for servers
    - they use CPU, memory, FDs, and ephemereal port space
- Takes up space in a table of your NAT box.
    - when you run out of space here, weird things happen

What should you do?

Optimizing for SPDY as a System Administrator

- Mind your cert chain
  - Use smaller, but complete certificate chains
  - Use wildcarts certs if they're available to you
  - You'll want to ask your cert provider questions about this
  - Unfortunately, this is a complicated issue

Optimizing for SPDY as a System Administrator

(cont'd)

- TCP settings
  - set your initcwnd to 10 packets worth
    - ip route change default via <ip> initcwnd 10
    - For more recent kernels, this is not necessary-- it is already the default
  - turn off tcp_slow_start_after_idle
    - sysctl -w net.ipv4.tcp_slow_start_after_idle=0

For System Administrators

(cont'd)

- DNS
  - When possible, be sure that your resolvers return the same set of IPs for any hostname that might match the same cert
  - Try to be sure that the set of IPs provided with any resolution only changes in ordering, not in content

```
If (new_domain_resolution.contains(IP of a pre-existing connection)
   && connection_SSL_cert.matches(new_domain))
        - Connection reuse! Profit!
```

For Application Developers:

- Don't shard hostnames
  - This might be required for HTTP, but not for SPDY
  - You'll still need to do it for HTTP, but, think about how to avoid doing it when the user is coming to you via SPDY

For Application Developers:

(cont'd)

- Plan on using server-push instead of inlining in the future
  - Server-pushed items are completely cacheable at the browser. This can make subsequent page renderings much faster.
  - Getting inlining right is non-trivial. What works great for this version of your site may make things worse with your next change.
    - Server-push with SPDY helps with this-- if you make a mistake, SPDY can help mitigate the damage

For Application Developers:

(cont'd)

- If your site's cert is a wildcard cert, use domains that match that cert instead of requiring others
  - e.g. google has a wildcard cert: "*.google.com"
  - If you use multiple domains, wildcard certs might end up being cheaper anyway...

On Standards

- SPDY has been developed as an open, publicly documented protocol.
- Source code for SPDY has been available as open-source since day 1.
- SPDY has been submitted to IETF as a proposal for HTTP/2.0

The Future of SPDY

We don't think we're done.

We'll be done only when we can serve a page in:
connection-setup + bytes/bandwidth time.

... We think we can get there! We'll need:

- name resolution push
- cert data push
- explicit proxy support

Browsers supporting SPDY

Today, there are two browsers that support SPDY and have it on by default:

- Chrome
- Firefox (V13)

This ends up being a whole lot of users.

Using or Supporting SPDY

There are a bunch of companies, organizations, and projects which use or support SPDY:

- Google's sites
  https://www.google.com, https://gmail.google.com, https://plus.google.com
- App Engine sites
  (if they use https)
- Twitter
  https://twitter.com
- Akamai
  (SPDY/2) technology preview late summer, part of the product line this Fall.
- Cotendo
  (http://www.akamai.com/cotendo)
- Nginx
  (http://mailman.nginx.org/pipermail/nginx-devel/2012-June/002343.html)
- (more coming on next slide)

Using or Supporting SPDY (cont'd)

There are a bunch of companies, organizations, and projects which use or support SPDY:

- F5's SPDY Gateway (http://www.f5.com/news-press-events/press/2012/20120508b.html)
- Strangeloop (http://www.strangeloopnetworks.com/products/overview/features/)
- Jetty (http://wiki.eclipse.org/Jetty/Feature/SPDY)
- mod_spdy (http://code.google.com/p/mod-spdy/)
- node-spdy (for node.js https://github.com/indutny/node-spdy)
- Momentum (https://github.com/jonasschneider/momentum)
- (more coming on next slide)

Using or Supporting SPDY (cont'd)

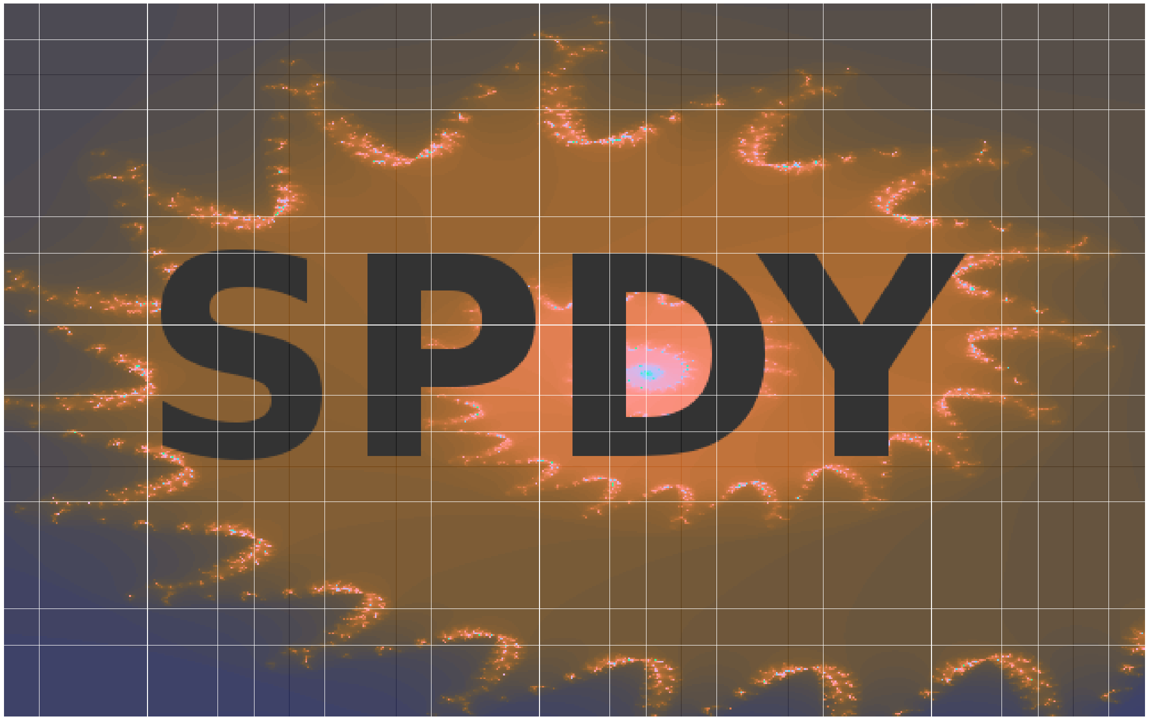
There are a bunch of companies, organizations, and projects which use or support SPDY:

- Amazon Silk
  (http://aws.amazon.com/amazonsilk-jobs/)
- Netty
  (http://netty.io/Blog/Netty+331+released+-+SPDY+Protocol+%21)
- SPDY-for-iPhone
  (https://github.com/sorced-jim/SPDY-for-iPhone)
- spdylay
  (https://github.com/tatsuhiro-t/spdylay)

- This is not a complete list!

A page to optimize

SPDY



The CSS

CSS

```
img {
float:left; margin:0px; padding:0px; height:20px; width:20px;
opacity:0.8; filter:alpha(opacity=80);
}
img:hover {
float:left; margin:0; padding:0;
opacity:1.0; filter:alpha(opacity=100);
}
```

The Javascript

JAVASCRIPT

```javascript
function main() {
  var up_thru_host = window.location.protocol + "//" + window.location.host;
  var pathname = window.location.pathname;
  var filebase = pathname.substr(0, pathname.lastIndexOf('/'));
  var tile_size_x = 20, tile_size_y = 20;
  var image_size_x = 640, image_size_y = 400;
  var rows = Math.ceil(image_size_y / tile_size_y);
  var columns = Math.ceil(image_size_x / tile_size_x);
  var text = [];
  for (x = 0; x < rows * columns; ++x) {
    if (x % columns == 0) text.push("<div style=\"clear:both;\"></div>");
    text.push("<img src='" + up_thru_host);
    if (window.location.protocol == "http:") text.push( ((x%8) + 1));
    text.push(filebase + "/s/" + x + ".png'/>");
  }
  document.getElementById("tiles_go_here").innerHTML = text.join('');
}
```

The HTML

HTML

```
<html>
<head>
  <title>SPDY SAMPLE</title>
  <style type="text/css">
            <-- CSS GOES HERE. -->
  </style>
  <script language=javascript>
            <-- JAVASCRIPT GOES HERE. -->
  </script>
</head>
<body>
  <div id="tiles_go_here"></div>
  <script language=javascript>
  main();
  </script>
</body> </html>
```

The PHP - main()

PHP

```php
function main() {
  $script_name = $_SERVER["SCRIPT_NAME"];
  $uri=substr($script_name, strrpos($script_name, '/') + 1);
  $server_name = $_SERVER["SERVER_NAME"];
  if ($fd = fopen($uri, "rb")) {
    $fn = $uri . ".push";
    $XAssociatedContent = makeXAssociatedContent($fn, "https://", $server_name);
    foreach ($XAssociatedContent as $line) header('x-associated-content: ' . $line, false);
    printFile($fd);
  } else {
    if ($uri == "index.html") {
      doPrintDirListing();
    } else {
      doFileNotFound();
    }
  }
}
```

The PHP - makeXAssociatedContent()

PHP

```php
function makeXAssociatedContent($filename, $scheme, $hostname) {
  $fp = fopen($filename, "r");
  $retarray = array();
  $retval = "";
  $xac_len = strlen('x-associated-content');
  $line_len = 0;
  if ($fp) {
    while ($line = fgets($fp)) {
      if (!$line) continue;
      if (($line_len += strlen($line)) >= 512 - $xac_len) {
        $retarray[] = $retval;
        $retval = ""; $line_len = 0;
      }
      $escaped_line = addslashes(str_replace("\n", '', $line));
      if ($retval) $retval = $retval . ',';
      $url = $scheme . $hostname . $escaped_line;
      $retval = $retval . '"' . $url . '"';
    }
  }
  if ($retval != "")
    $retarray[] = $retval;
  return $retarray;
}
```

The PHP - the rest

PHP

```
function printFile($fd) {
  while (!feof($fd)) {
    $read_buf = fread($fd, 4096);
    print $read_buf;
  }
}

function doFileNotFound() {
  header("status: 404");
  header("HTTP/1.0 404 Not Found");
  print "<html><head></head><body>\n";
  print "Sorry, file:" . $uri . "wasn't found.\n";
  print "</body></html>";
}

function doPrintDirListing() {
  print "<html><head></head><body>";
  $files = scandir(".");
  foreach ($files as $file) {
    print '<a href="' . $file . '">' . $file . "</a><br>\n";
  }
  print "</body></html>";
}
```

Bits O Apache Config

/etc/apache2/sites-available/default-ssl

```
    ...
    <Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride All
        Order allow,deny
        allow from all
    </Directory>
    AliasMatch "^/.*(.html)$"  /var/www/push_mapper.php
    ...
```

/etc/apache2/mods-available/spdy.conf

```
<IfModule spdy_module>
    SpdyEnabled on
    SpdyMaxThreadsPerProcess 100
    SpdyMaxStreamsPerConnection 1000
</IfModule>
```

The .push file

spdy_push_middle_sample.html.push

```
/s/228.png
/s/229.png
/s/230.png
/s/231.png
/s/232.png
/s/234.png
/s/235.png
/s/236.png
/s/237.png
/s/238.png
/s/239.png
/s/240.png
/s/241.png
/s/242.png
/s/243.png
/s/244.png
/s/245.png
/s/246.png
/s/247.png
/s/248.png
/s/249.png
/s/250.png
...
```

<Thank You!>

fenix@google.com

Google
Developers