How speedy is SPDY?

Xiao (Sophia) Wang
University of Washington
Facebook Inc.
HTTP/1.1: The standard to load Web pages becomes slow for rich, modern pages.

Google developed **SPDY** to make the Web faster - Being deployed - Basis for HTTP/2.0

How much better is SPDY than HTTP?
HTTP/1.1 problems
HTTP/1.1 problems

- Opens too many TCP connections
HTTP/1.1 problems

- Opens too many TCP connections
- Initiates object transfers strictly by the client
HTTP/1.1 problems

- Opens too many TCP connections
- Initiates object transfers strictly by the client
- Compresses only HTTP payloads, not headers
HTTP/1.1 problems

SPDY is proposed to address these issues

• Opens too many TCP connections
• Initiates object transfers strictly by the client
• Compresses only HTTP payloads, not headers
SPDY

- Opens too many TCP connections
- Multiplexes sliced frames into a single TCP connection
SPDY

- Opens too many TCP connections
- Multiplexes sliced frames into a single TCP connection
- Prioritizes Web objects
Client

SPDY

• Initiates object transfers strictly by the client
• Allows servers to initiate Web object transfers

Server
SPDY

- Compresses only HTTP payloads, not headers
- Compresses both HTTP payloads and headers
How well does SPDY perform?

Google

SPDY helps 27% to 60%
How well does SPDY perform?

SPDY helps 27% to 60%.

SPDY sometimes helps and sometimes hurts.

Measurement results conflict.
Goals

• A systematic study of SPDY that
  – Extensively sweeps the parameter space
  – Links SPDY performance to underlying factors
  – Identifies the dominant factors
Many factors external to SPDY affect SPDY

Approach
Isolate factors, sweep the parameter space

- Network parameters
- TCP settings
- Web page effects

- RTT
- Bandwidth
- Loss rate

- TCP initial congestion window
- Synthetic objects
- Real objects
- Real pages

Challenge
Challenge

Page load time has high variance

Variance: 0.5 second
Difference: 0.02 second

Approach

Control source of variability by
- Experimenting in a **controlled network**
- Using **our emulator** instead of browsers
Dependencies between network and browser computation affect page loads

**Challenge**

No browser

Browser computation

Elapsed time

Objects

Elapsed time

Objects
Dependencies between network and browser computation affect page loads.

**Challenge**

Preserve dependencies.

**Approach**

- No browser
- Browser computation

Elapsed time

Objects
Outline

• Understanding SPDY’s performance with
  – Synthetic objects
  – Real objects
  – Real pages

• Enhanced policies for SPDY
Outline

• Understanding SPDY’s performance with
  – Synthetic objects
  – Real objects
  – Real pages
• Enhanced policies for SPDY
Extensively sweep parameter space

<table>
<thead>
<tr>
<th>Factors</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTT</td>
<td>20ms, 100ms, 200ms</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>1Mbps, 10Mbps</td>
</tr>
<tr>
<td>Loss rate</td>
<td>0, .5%, 1%, 2%</td>
</tr>
<tr>
<td>TCP IW</td>
<td>3, 10, 21, 32</td>
</tr>
<tr>
<td>Web obj. size</td>
<td>100B, 1K, 10K, 100K, 1M</td>
</tr>
<tr>
<td># of objects</td>
<td>2, 8, 16, 32, 64, 128, 512</td>
</tr>
</tbody>
</table>

Make HTTP requests
Link SPDY performance to factors

→ Decision tree analysis

Six factors,
Thousands of data points
SPDY helps on small objects

Explanation
Unlike in HTTP, a TCP segment can carry multiple Web objects in SPDY.

Why SPDY helps
SPDY helps on large objects, low loss

In HTTP, Multiple connections compete with each other

⇒ More retransmissions
SPDY hurts on large objects, high loss

Most performance impact of SPDY comes from a single TCP connection.
Identify dominant factors

- # obj
- BW
- RTT
- obj size
- loss
- IW

Importance?
Identify dominant factors

- obj size
- loss
- # obj

more important than

- RTT
- BW
- IW

# obj shows a trend

TCP IW

RTT: 200ms
BW: 10Mbps
Loss: 0
IW: 3
obj size: 10KB
# obj: 8

# obj shows a trend

IW doesn’t show a trend
Does SPDY help stragglers?

- In our experiments, we find that SPDY helps little for stragglers.

This hypothesis is weak since it only argues with cwnd.
Outline

• Understanding SPDY’s performance with
  – Synthetic objects
  – Real objects
  – Real pages

• Enhanced policies for SPDY
# Synthetic objects ➔ Real objects

<table>
<thead>
<tr>
<th>Factors</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network parameters</strong></td>
<td></td>
</tr>
<tr>
<td>RTT</td>
<td>20ms, 100ms, 200ms</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>1Mbps, 10Mbps</td>
</tr>
<tr>
<td>Loss rate</td>
<td>0, .5%, 1%, 2%</td>
</tr>
<tr>
<td><strong>TCP settings</strong></td>
<td></td>
</tr>
<tr>
<td>TCP IW</td>
<td>3, 10, 21, 32</td>
</tr>
<tr>
<td><strong>Web objects</strong></td>
<td></td>
</tr>
<tr>
<td>Web obj. size</td>
<td>Top 200 Alexa pages</td>
</tr>
<tr>
<td># of objects</td>
<td></td>
</tr>
</tbody>
</table>

Make HTTP requests
SPDY helps 60% in the median case because it largely reduces retransmissions.
Outline

• Understanding SPDY’s performance with
  – Synthetic objects
  – Real objects
  – Real pages

• Enhanced policies for SPDY

Browser effects
Assumption that objects are fetched at the same time does not hold.
Epload captures browser effects

• Recorder: capture the dependency graph
• Replayer: make network requests while simulating the computation portions

Epload makes experiments **reproducible**
Real objects $\rightarrow$ Real pages

<table>
<thead>
<tr>
<th>Factors</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network parameters</td>
<td></td>
</tr>
<tr>
<td>RTT</td>
<td>20ms, 100ms, 200ms</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>1Mbps, 10Mbps</td>
</tr>
<tr>
<td>Loss rate</td>
<td>0, .5%, 1%, 2%</td>
</tr>
<tr>
<td>TCP settings</td>
<td></td>
</tr>
<tr>
<td>TCP IW</td>
<td>3, 10, 21, 32</td>
</tr>
<tr>
<td>Web objects</td>
<td></td>
</tr>
<tr>
<td>Web obj. size</td>
<td>Top 200 Alexa pages</td>
</tr>
<tr>
<td># of objects</td>
<td></td>
</tr>
</tbody>
</table>

Emulate page loads with Epload
SPDY helps marginally because
• Computation and dependencies

Dependencies and computation in real page loads reduce the impact of SPDY.
Outline

• Understanding SPDY’s performance with
  – Synthetic objects
  – Real objects
  – Real pages

• Enhanced policies for SPDY
Improving SPDY with server push

• Leverage information from dependency graphs
  – Web objects that are closer to the root should be pushed earlier
Improving SPDY with server push

Mod_spdy’s: one level of HTML embedding

Our policy: one level of the dependency graph
Improving SPDY with server push

• Server push with our policy and mod_spdy’s both helps page load time by **10%~30%**
• Our server push policy reduces **80%** of pushed bytes compared to mod_spdy’s

Improving page load performance requires restructuring the page load process, e.g. server push.
Other experiments in the paper

• With domain sharding
  – Compared domain sharding policies
    • One conn. for the whole page
    • One conn. for a second-level domain (SLD)
    • One conn. for a domain
  – Per-SLD policy is comparable to per-page; per-domain policy hurts performance

• With SSL/TLS
  – Tested SPDY and HTTP over SSL/TLS
  – Larger latencies but same conclusions
Conclusions

• We experimented with SPDY page loads over a large parameter space
• Most performance impact of SPDY over HTTP comes from its single TCP connection
• Browser computation and dependencies in real pages reduce the impact of SPDY
• To improve further, we need to restructure the page load process
Data

We release the data obtained by sweeping the parameter space and welcome further analysis on this data. Here is our setting.

Download all data (211KB) (downloaded 3 times)

We tabularize our data below and allow sort by column. We provide plots that show trends in one parameter by fixing the other parameters. Guide on how to plot trends. To download the network trace of a data point, just click on the link to the PLT (page load time) of that data point.