How Speedy is SPDY?

Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy and David Wetherall
University of Washington

Presented by: Dipendra Kumar Jha
HTTP vs SPDY

- HTTP/1.1: The standard to load Web pages
- HTTP/1.1 becomes slow for rich, modern pages
- How much better is SPDY than HTTP?
  - Google developed SPDY to make the Web faster
  - SPDY started to be deployed (Facebook, Google)
  - Browsers also support SPDY (Chrome, Firefox, IE11)
HTTP vs SPDY

**HTTP/1.1 Problems**

- Opens too many TCP connections
- Initiates object transfers strictly by the client
- A TCP segment cannot carry more than one HTTP request or response
- Compresses only HTTP payloads, not headers

**SPDY is proposed to address these issues**

- Multiplexes sliced frames into a single TCP connection
- Prioritizes Web objects
- Allows servers to initiate Web object transfers (Server Push)
- Compresses both HTTP payloads and headers (Header compression)
How well does SPDY perform?

**Google**
SPDY helps 27% to 60%

**Microsoft**
SPDY sometimes helps and sometimes hurts. Overall, SPDY helps < 10%.
Goals

A systematic study of SPDY that:

- Extensively sweeps the parameter space
- Links SPDY performance to underlying factors
- Identifies the dominant factors
Challenges in study of SPDY

Challenge

Many factors external to SPDY affect SPDY

Approach

Isolate factors, sweep the parameter space

- Network Parameters (RTT, Bandwidth, Loss rate)
- TCP settings (TCP initial congestion window)
- Web page effects (Synthetic objects, Real objects, Real pages)
Challenges in study of SPDY

Challenge
Page load time has high variance

Approach
Control source of variability by

- Experimenting in a controlled network
- Using Epload instead of real browsers
Challenges in study of SPDY

Challenge

Dependencies between network and browser computation affect page loads

Approach

Preserve dependencies
Experimenting with synthetic pages

<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
Experimenting with synthetic pages

The decision tree analysis generates the likelihood that a configuration works better under SPDY (or HTTP). If this likelihood is over 0.75, we mark the branch as SPDY better; a leaf pointing to EQUAL means SPDY helps; a leaf pointing to SPDY (HTTP) means SPDY (HTTP) helps. A leaf pointing to SPDY (HTTP) means SPDY (HTTP) helps. A leaf pointing to EQUAL means SPDY helps.

Here, we sort out the most important findings, meaning how much hypotheses contribute to SPDY performance. There have been many hypotheses as to whether SPDY improves performance under certain conditions, but degrades performance under other conditions. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance.

To understand the conditions under which SPDY outperforms HTTP (or HTTP), we say that SPDY and HTTP perform equally.

In the analysis, each configuration is a combination of values for all factors listed based on decision tree analysis. In the analysis, each configuration is a combination of values for all factors listed based on decision tree analysis. In the analysis, each configuration is a combination of values for all factors listed based on decision tree analysis.

To populate the branches, we also generate supplemental decision trees based on subsets of factors. Each supplemental mental decision tree has a prediction accuracy of 84% or higher. Last, we merge the branches from supplemental decision trees into the original decision tree.

The decision tree also depicts the relative importance of contributing factors. Intuitively, factors close to the root of the decision tree affect SPDY performance more than those near the leaves. This is because the decision tree places the important factors near the root to reduce the number of branches. We find that object size and loss play a less important role. Many large objects under low loss.

The decision tree also depicts the relative importance of contributing factors. Intuitively, factors close to the root of the decision tree affect SPDY performance more than those near the leaves. This is because the decision tree places the important factors near the root to reduce the number of branches. We find that object size and loss play a less important role. Many large objects under low loss.

We obtain the decision tree in Figure 2 as follows.

- First, we produce a decision tree based on all the factors.
- To understand the conditions under which SPDY outperforms HTTP (or HTTP), we say that SPDY and HTTP perform equally.
- If this likelihood is over 0.75, we mark the branch as SPDY better or equal to HTTP; otherwise, we say that SPDY and HTTP perform equally.

There have been many hypotheses as to whether SPDY improves performance under certain conditions, but degrades performance under other conditions. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance.

The decision tree analysis generates the likelihood that a configuration works better under SPDY (or HTTP). If this likelihood is over 0.75, we mark the branch as SPDY better; a leaf pointing to EQUAL means SPDY helps; a leaf pointing to SPDY (HTTP) means SPDY (HTTP) helps. A leaf pointing to SPDY (HTTP) means SPDY (HTTP) helps. A leaf pointing to EQUAL means SPDY helps.

Here, we sort out the most important findings, meaning how much hypotheses contribute to SPDY performance. There have been many hypotheses as to whether SPDY improves performance under certain conditions, but degrades performance under other conditions. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance.

To understand the conditions under which SPDY outperforms HTTP (or HTTP), we say that SPDY and HTTP perform equally.

In the analysis, each configuration is a combination of values for all factors listed based on decision tree analysis. In the analysis, each configuration is a combination of values for all factors listed based on decision tree analysis. In the analysis, each configuration is a combination of values for all factors listed based on decision tree analysis.

To populate the branches, we also generate supplemental decision trees based on subsets of factors. Each supplemental mental decision tree has a prediction accuracy of 84% or higher. Last, we merge the branches from supplemental decision trees into the original decision tree.

The decision tree also depicts the relative importance of contributing factors. Intuitively, factors close to the root of the decision tree affect SPDY performance more than those near the leaves. This is because the decision tree places the important factors near the root to reduce the number of branches. We find that object size and loss play a less important role. Many large objects under low loss.

The decision tree also depicts the relative importance of contributing factors. Intuitively, factors close to the root of the decision tree affect SPDY performance more than those near the leaves. This is because the decision tree places the important factors near the root to reduce the number of branches. We find that object size and loss play a less important role. Many large objects under low loss.

We obtain the decision tree in Figure 2 as follows.

- First, we produce a decision tree based on all the factors.
- To understand the conditions under which SPDY outperforms HTTP (or HTTP), we say that SPDY and HTTP perform equally.
- If this likelihood is over 0.75, we mark the branch as SPDY better or equal to HTTP; otherwise, we say that SPDY and HTTP perform equally.

There have been many hypotheses as to whether SPDY improves performance under certain conditions, but degrades performance under other conditions. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance.

The decision tree analysis generates the likelihood that a configuration works better under SPDY (or HTTP). If this likelihood is over 0.75, we mark the branch as SPDY better; a leaf pointing to EQUAL means SPDY helps; a leaf pointing to SPDY (HTTP) means SPDY (HTTP) helps. A leaf pointing to SPDY (HTTP) means SPDY (HTTP) helps. A leaf pointing to EQUAL means SPDY helps.

Here, we sort out the most important findings, meaning how much hypotheses contribute to SPDY performance. There have been many hypotheses as to whether SPDY improves performance under certain conditions, but degrades performance under other conditions. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance. However, RTT, bandwidth, and TCP initial rate are the most important factors in predicting SPDY performance.
Why does SPDY help or hurt?

- SPDY helps on small objects
  - TCP implements congestion control by counting outstanding packets not bytes

- SPDY benefits from having a single connection
  - Fewer retransmission, less TCP connection setup time, less time pipe is idle

- SPDY degrades under high loss due to the use of a single pipe
  - Aggressively reduces the congestion window
Experimenting with Real objects

<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>TCP IW</td>
</tr>
<tr>
<td></td>
<td>3, 10, 21, 32</td>
</tr>
<tr>
<td>Web objects</td>
<td>Web obj. size</td>
</tr>
<tr>
<td></td>
<td>Top 200 Alexa pages</td>
</tr>
<tr>
<td></td>
<td># of objects</td>
</tr>
</tbody>
</table>

Make HTTP requests
Experimenting with Real objects

SPDY helps 60% in the median case because it largely reduces retransmissions.
TCP modifications: TCP+

Figure 6: SPDY performance across 200 pages with object sizes and numbers of objects drawn from real pages. SPDY helps more under a 1Mbps bandwidth.

Figure 7: SPDY helps reduce retransmissions.

4.4 TCP modifications

Previously, we found that SPDY hurts mainly under high packet loss because a single TCP connection reduces the congestion window more aggressively than HTTP's parallel connections. Here, we demonstrate that the negative impact can be mitigated by simple TCP modifications.

Our modification (a.k.a., TCP+) mimics behaviors of concurrent connections with a single connection. Let the number of parallel TCP connections be $n$. First, we propose to multiply the initial window by $n$ to reduce the effect of slow start. Second, we suggest scaling the receive window by $n$ to ensure that the SPDY connection has the same amount of receive buffer as HTTP's parallel connections. Third, when packet loss occurs, the congestion window ($cwnd$) backs off with a rate $\beta' = 1 - (1 - \beta) / n$ where $\beta$ is the original backoff rate. In practice, the number of concurrent connections changes over time. Because we are unable to pass this value to the Linux kernel in real time, we assume that HTTP uses six connections and set $n = 6$. We use six here because it is found optimal and used by major browsers [17].

We perform the same set of SPDY experiments with both synthetic and real pages using TCP+. Figure 8 shows that SPDY performs better with TCP+, and the decision tree analysis for TCP+ suggests that loss rate is no longer a key factor that determines SPDY performance.

To evaluate the potential side effects of TCP+, we look at the number of retransmissions produced by TCP+. Figure 9 shows that SPDY still produces much fewer retransmissions with TCP+ than with HTTP, meaning that TCP+ does not abuse the congestion window under the conditions that we experimented with. Here, we aim to demonstrate that SPDY's negative impact under high random loss can be mitigated by tuning the congestion window. Because the loss patterns in real networks are likely more complex, a solution for real networks requires further consideration and extensive evaluations and is out of the scope of this paper.

5 Web pages and SPDY

This section examines how SPDY performs for real Web pages. Real page loads incur dependencies and computation that may affect SPDY's performance. To incorporate dependencies and computation while controlling variability, we develop a page load emulator $E_{\text{load}}$.
## Web pages and SPDY

<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>Top 200 Alexa pages</td>
</tr>
<tr>
<td># of objects</td>
<td></td>
</tr>
</tbody>
</table>

Emulate page loads with **Epload**
Web pages and SPDY

Figure 12: SPDY performance using emulated page loads. Compared to Figure 6, it suggests that dependencies and computation reduce the impact of SPDY and that RTT and bandwidth become more important.

Obtain the dependency graphs of the top 200 Alexa Web pages [1]. Epload runs on a Mac with 2GHz dual core CPU and 4GB memory. We vary other factors based on the parameter space described in Table 1. Due to space limit, we only show figures under a 10Mbps bandwidth.

Figure 12 shows the performance of SPDY versus HTTP after incorporating dependencies and computation. Compared to Figure 6, dependencies and computation largely reduce the amount that SPDY helps or hurts. We make the following observations along with supporting evidence. First, computation and dependencies increase PLTs of both HTTP and SPDY, reducing the network load. Second, SPDY reduces the amount of time a connection is idle, lowering the possibility of slow start (see Figure 13). Third, dependencies help HTTP by making traffic less bursty, resulting in fewer retransmissions (see Figure 14). Fourth, having fewer outstanding objects diminishes SPDY's gains, because SPDY helps more when there are a large number of outstanding objects (as suggested by the decision tree in Figure 2).

Here, we see that dependencies and computation reduce and can easily nullify the benefits of SPDY, implying that speeding up computation or breaking dependencies might be necessary to improve the PLT using SPDY. Interestingly, we find that RTT and bandwidth now play a more important role in the performance of SPDY. For example, Figure 12 shows that SPDY helps up to 80% of the pages under low bandwidths, but only 55% of the pages under high bandwidths. This is because RTT and bandwidth determine the amount of time page loads spend in network relative to computation, and further the amount of impact that computation has on SPDY. This explains why SPDY provides minimal improvements under good network conditions (see Figure 12(c)).

To identify the impact of computation, we scale the time spent in each computation activity by factors of 0, 0.5, and 2. Figure 15 shows the performance of SPDY versus HTTP, both with scaled computation and under high bandwidths, suggesting that speeding up computation increases the impact of SPDY. Surprisingly, speeding up computation to the extreme is sometimes no better than a x2 speedup. This is because computation delays the requesting of dependent objects which allows for previously requested objects to be loaded faster, and therefore possibly lowers the PLT.

5.3 Advancing SPDY

SPDY provides two mechanisms, i) prioritization and ii) server push, to mitigate the negative effects of dependencies and computation of real page loads. However, little is known about how to better use the mechanisms. In this section, we explore advanced policies to speed up page loads.
Web pages and SPDY

Dependencies and computation in real page loads reduce the impact of SPDY
Server push

5.3.3 Server push

SPDY allows servers to push objects to save round trips. However, server push is non-trivial because there is a tension between making page loads faster and wasting bandwidth. Particularly, one should not overuse server push if pushed objects are already cached. Thus, the key goal is to speed up page loads while keeping the cost low.

We find no standard or best practices guidance from Google on how to do server push. Mod spdy can be configured to push up to an embedding level, which is defined as follows: the root HTML page is at embedding level 0; objects at embedding level $i$ are those whose URLs are embedded in objects at embedding level $i - 1$.

An alternative policy is to push based on the depth information.

Figure 18 shows server push performance (i.e., push all objects, one embedding level, and one dependency level) compared to baseline SPDY. We find that server push helps, especially under high RTT. We also find that pushing by dependency incurs comparable speedups to pushing by embedding, while benefiting from an 80% reduction in pushed bytes (Figure 18(a)). Note that server push does not always help because pushed objects share bandwidth with more important objects. In contrast to prioritization, server push can help because it breaks dependencies which limits the performance gains of SPDY.

5.4 Putting it all together

We now pool together the various enhancements (i.e., TCP+ and server push by one dependency level). Figure 19 shows that this improves SPDY by 30% under high RTTs. But this improvement largely diminishes under low RTTs where computation dominates page load times.

6 Discussions

SPDY in the wild:

To evaluate SPDY in the wild, we place clients at Virginia (US-East), North California (US-West), and Ireland (Europe) using Amazon EC2 micro-instances. We add explanatory power by periodically probing network parameters between clients and the server, and find that RTTs are consistent: 22ms (US-East), 71ms (US-West), and 168ms (Europe). For all vantage points, bandwidths are high (10Mbps to 143Mbps) and loss rates are extremely low. These network parameters well explain our SPDY evaluations in the wild (not shown due to space limit) that are similar to synthetic ones under high bandwidths and low loss rates. The evaluations here are preliminary and covering a complete set of scenarios would be future work.

Domain sharding:

As suggested by SPDY best practices [18], we used a single connection to fetch all the objects of a page to eliminate the negative impact of domain sharing. In practice, migrating objects to one domain suffers from deployment issues given popular uses of third parties (e.g., CDNs, Ads, and Analytics). To this
Server push

5.3.3 Server push

SPDY allows servers to push objects to save round trips. However, server push is non-trivial because there is a tension between making page loads faster and wasting bandwidth. Particularly, one should not overuse server push if pushed objects are already cached. Thus, the key goal is to speed up page loads while keeping the cost low.

We find no standard or best practices guidance from Google on how to do server push. Mod spdy can be configured to push up to an embedding level \(i\), which is defined as follows: the root HTML page is at embedding level 0; objects at embedding level \(i\) are those whose URLs are embedded in objects at embedding level \(i-1\).

An alternative policy is to push based on the depth information.

Figure 18 shows server push performance (i.e., push all objects, one embedding level, and one dependency level) compared to baseline SPDY. We find that server push helps, especially under high RTT. We also find that pushing by dependency incurs comparable speedups to pushing by embedding, while benefiting from an 80% reduction in pushed bytes (Figure 18(a)). Note that server push does not always help because pushed objects share bandwidth with more important objects. In contrast to prioritization, server push can help because it breaks dependencies which limits the performance gains of SPDY.

5.4 Putting it all together

We now pool together the various enhancements (i.e., TCP+ and server push by one dependency level). Figure 19 shows that this improves SPDY by 30% under high RTTs. But this improvement largely diminishes under low RTTs where computation dominates page load times.

6 Discussions

SPDY in the wild:

To evaluate SPDY in the wild, we place clients at Virginia (US-East), North California (US-West), and Ireland (Europe) using Amazon EC2 micro-instances. We add explanatory power by periodically probing network parameters between clients and the server, and find that RTTs are consistent: 22ms (US-East), 71ms (US-West), and 168ms (Europe). For all vantage points, bandwidths are high (10Mbps to 143Mbps) and loss rates are extremely low. These network parameters well explain our SPDY evaluations in the wild (not shown due to space limit) that are similar to synthetic ones under high bandwidths and low loss rates. The evaluations here are preliminary and covering a complete set of scenarios would be future work.

Domain sharding:

As suggested by SPDY best practices [18], we used a single connection to fetch all the objects of a page to eliminate the negative impact of domain sharing. In practice, migrating objects to one domain suffers from deployment issues given popular uses of third parties (e.g., CDNs, Ads, and Analytics).
Conclusions

• 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 dependences in real pages reduce the impact of SPDY

• To improve further, we need to restructure the page load process
Thank You

Questions??
Experimenting with synthetic pages – dominant factors

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

# Obj shows a trend

TCP IW

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

IOW doesn’t show a trend