SPDY and What to Consider for HTTP/2.0

mike belshe
Why am I here?

SPDY started over 3 years ago

Reduced latency is now proven

It's better for the network

Let's focus on interoperability
Who is using SPDY?

- Google Chrome & All Google Web Properties
- Mozilla Firefox
- Twitter
- Amazon Silk

Others: Cotendo, Strangelope, iPhone client, Apache mod-spdy, nginx beta, jetty, netty, libraries in python, node.js, erlang, ruby, go, and C
How did SPDY come to be?

wanted reduced web page latency *for users*
What SPDY is Not

A transport layer protocol (like TCP)

Rocket Science

Cheap Compression Tricks
What SPDY is

An amalgam of well-known ideas based on performance data:

- multiplexing
- prioritization
- compression
- server push
- transparent to HTTP app servers
deployable today
Real deployment has shown also

- Better for the network
- Better for Mobile
  HTTP is not just for HTML
  Battery life matters
Background: What is a WebPage?

- 86 resources
- 13 hosts
- 800+KB
- only 66% compressed (top sites are ~90% compressed)
Background: Poor Network Utilization

![Effective Bandwidth of HTTP](image)

The graph illustrates the effective bandwidth of HTTP for different bandwidth levels. As the bandwidth increases from 1Mbps to 10Mbps, the effective bandwidth also increases, with 10Mbps showing the highest effective bandwidth.
More Bandwidth Doesn't Help
But Reducing Round Trip Time Does
Background: HTTP Connections

2010: Average 29 connections per page.
SPDY Features
1. Multiplexing

- Small, fixed length frames
- Fully interleaved streams
- Streams can be created by either endpoint with zero round trips.
- Many implementors have remarked it's easy to implement!
Before Multiplexing
After Multiplexing
2. Prioritization

- Not all requests are equal!
- Failure to prioritize is actually slower
- Must consider two metrics:
  - Time to first render
  - Overall Page Load Time
- SPDY allows client-specified priorities with server best effort to deliver
3. Header Compression

- SPDY uses stateful compression across requests
- Using zlib, achieves 85-90% compression
- Don't care if compressor is zlib; only care about session state.
- Must be mandatory
Compression Savings

Time Saved

45-1142ms per Page Load

Yahoo | Google | Cnet | Amazon

Uplink @512Kbps | Downlink @2Mbps
Better Networking
SPDY "Less is More" Connections

Connections Per Web Page

HTTP

SPDY

# of connections
SPDY "Less is More"
Uplink data

SPDY vs HTTP Upload KB Sent (Top-45 pages)

HTTP

SPDY

51% reduction
SPDY "Less is More"
Downlink data

SPDY vs HTTP Download KB (Top-45 pages)

HTTP

SPDY

4% reduction
SPDY "Less is More"
Total Packets

SPDY vs HTTP Total Packets (Top-45 pages)

HTTP

SPDY

19% reduction
Performance Results
Google Results

SPDY vs HTTPS Chrome 12, Mar 22-April 5

SPDY

HTTPS

15.4% improvement
Cotendo Tests

Amazon.com home

HTTP

3G AT&T

~200ms RTT

PLT: 12.50 secs
Cotendo Tests

Amazon.com home

SPDY

3G AT&T

~200ms RTT

PLT: 6.26 secs

-49%
Other results

- Firefox confirmed Chrome results
- Google recently reported that SPDY over SSL is now faster than HTTP without SSL
- BoostEdge paper confirms Google numbers
- *need vendors to publish more!*
Deployment
A Process of Elimination

- Transport choices: TCP or UDP
  - Chose TCP

- Port choices: 80 or 443
  - But both are taken!

- Chrome test shows usability of port 80 for non HTTP protocols is <75%.
  - Using port 80 makes SPDY like Pipelining.

- Port 443 is the only untampered port.

- Other ports: blocked by firewalls
Pause - That was the Big Picture

"Better is the enemy of good"

- The aforementioned items are the non-controversial parts of SPDY.
- HTTP/2.0 should take those concepts.
- Minutiae doesn't matter:
  - exact framing syntax
  - exact compression algorithm
- Stay Focused on the Big Picture!
Why Nots?
Why not SCTP?

Multiplexing over a single TCP stream does have one element of head-of-line blocking.

But SCTP has problems:
- Not available on most platforms
- Requires administrative privs to install (so it can't be bundled easily with browser installs)
- Incompatible with NAT on today's internet.
Why not Pipelining?

Pipelining was introduced a decade ago.

- Wasn't deployable due to intermediaries that didn't handle it properly.
- It has complex head-of-line blocking problems (hanging GETs)
- Firefox team list of heuristics is huge. SPDY was easier to build than pipelining.
- Counterpoint: mobile uses pipelining. Does it work?
Why 1 Connection?

- More efficient for network, memory use, and server scalability; better compression.
- Don't have to wait for a handshake to complete before sending a request.
- Doesn't encourage Buffer bloat. (Jim Gettys)
- Lets the transport do what it does best.
- Would like to see more research here.
SPDY for Mobile
Mobile is Different

- New client-side problems
  - Battery life constraints
  - Small CPUs (changing fast!)

- New Network Properties
  - Latency from 150 - 300ms per Round Trip
  - Bandwidth 1-4Mbps

- New use cases
  - Mobile Web Browsers are 1st generation
    - So web browsing sucks
  - Everyone uses Apps w/ REST APIs anyway
SPDY and Mobile

- Fewer connections/bytes/packets reduces transmit requirements of radio

- Mobile connection management is different due to NAT and in-and-out networks.
  - Can't use TCP Keepalives
  - PING frame detects closed connections quickly

- Header compression minimizes upstream sends

- 1 conn per domain minimizes tcp-level control traffic
The Tough Stuff
Don't make things "optional"

- Optional features are disabled features.  
  e.g. pipelining.

- Optional features are buggy.  
  e.g. absolute URIs fail on many HTTP/1.1 servers.

- Feature detection often takes a round-trip.  
  e.g. does it support a compressed request?

- Proxies will tamper with option negotiation.  
  e.g. Accept-EnXcoding
Security

I often hear that security is difficult/expensive/costly or unwanted.

I've NEVER heard this complaint from a user.

I've ONLY heard this complaint from proxy and server implementors.

Could it be that users just expect it to be secure?
What Security Can HTTP/2.0 Provide?

- Security is accomplished across the stack, not at a single layer. But HTTP does play a role.

- Requiring SSL with HTTP/2.0 will:
  - Protect the user from eavesdroppers (firesheep!)
  - Protect from content tampering
  - Protect the protocol for future extensions
  - Authenticate servers
Insecure Protocols Hurt Users

Without integrity & privacy, you enable anyone to:

- record data about you
- inject advertisements into your content
- prevent access to certain sites
- alter site content
- limit your bandwidth (for any reason)

Is this what the user wants?
Insecure Protocols Enable Transparent Proxies

- Transparent proxies are proxies that you didn't opt-in to
  - As a site operator, they can alter your content
  - As a user, they can alter your web experience

- Transparent proxies are to blame for many of our protocol woes:
  - Inability to fix HTTP/1.1 pipelining
  - Turning off compression behind the user's back

- They are easy to deploy, however...
SSL is not Expensive

- Twitter and Google rolled out with zero additional hardware.
- Bulk encryption (RC4) is basically free.
- Handshakes are a little expensive, but <1% of CPU costs.
- Certificates are free.
- SPDY + SSL is faster than HTTP.
Is an insecure protocol legal anymore?

- Privacy laws in the US & EU make those that leak private information liable for the losses

- Should web site administrators need to know how HTTP works in order to obey basic laws?
Recognizing Different Requirements

We need multiple protocols, not options
We have distinct use cases

- **End User HTTP**
  - targets consumers and Internet User needs

- **BackOffice HTTP**
  - for those using HTTP in behind their own firewalls

- **Caching HTTP (also corp firewall HTTP)**
  - For corporate environments or organizations sharing a common cache
  - May not be a separate protocol, but lets make it work explicitly.
End User HTTP

- Optimized for the Internet Consumer.

- Features:
  - Always secure (safe to use in the Cafe)
  - Always compressed
  - Always fast
BackOffice HTTP

- Used for backoffice server infrastructure, already behind your own firewalls.

- Features:
  - Not implemented by browsers
  - Makes SSL optional
  - Makes Compression optional
Caching HTTP

● Used by corporations with filtering firewalls or those that want to have an external cache

● Features:
  ○ User opts-in. Never transparent.
  ○ SSL to the proxy; proxy brokers the request to origin
  ○ Respects HSTS
  ○ Reduces need for SSL MITM
Thank you!

Looking forward to a fantastic HTTP/2.0!