DOI:10.1145/2380656.2380673

Improved performance and a proven deployment strategy make SPDY a potential successor to HTTP.

BY BRYCE THOMAS, RAJA JURDAK, AND IAN ATKINSON

SPDYing Up the Web

TODAY'S WEB BEARS little resemblance to the Web of a decade ago. A Web page today encapsulates tens to hundreds of resources pulled from multiple domains. JavaScript is the technological staple of Web applications, not a tool for frivolous animation. Users access the Web from diverse device form factors, while browsers have improved dramatically. A constant throughout this evolution is the underlying application-layer protocol—HTTP—providing fertile ground for Web growth and evolution but was designed at a time of far less page complexity. Moreover, HTTP is not optimal, with pages taking longer to load. Studies over the past five years suggest even 100 milliseconds additional delay can have a quantifiably negative effect on Web use,[9] spurring interest in improving Web performance. One such effort is SPDY, a potential successor to HTTP championed by Google that requires both client and server changes, a formidable hurdle to widespread adoption. However, early client support from major browsers Chrome and Firefox suggests SPDY is a protocol being taken seriously. Though server support for SPDY is growing through projects like mod_spdy[12] truly widespread server adoption is likely to take time. In the interim, SPDY gateways are emerging as a compelling transition strategy, promising to accelerate SPDY adoption by functioning as a translator between SPDY-enabled clients and non-SPDY-enabled servers (see Figure 1). A variety of incentives motivate organizations to deploy and users to adopt SPDY gateways, as described here.

SPDY (pronounced SPeeDY) is an experimental low-latency application-layer protocol[27] designed by Google and introduced in 2009 as a drop-in replacement for HTTP on clients and servers. SPDY retains the semantics of HTTP, allowing content to remain unchanged on servers while adding request multiplexing and prioritization, header compression, and server push of resources. Since 2009, SPDY has undergone metamorphosis from press release written and distributed by Google to production protocol implemented by some of the highest-profile players on the Web.

Figure 2 is a timeline of SPDY milestones, first appearing publicly in a

» key insights

- **SPDY seeks to improve Web page load times by making fewer round trips to the server.**
- **Client-side browser support for SPDY has grown rapidly since 2009, and SPDY gateways offer a transition strategy that does not rely on server-side support.**
- **Open SPDY gateways are an opportunity for organizations to capitalize on the behavioral browsing data they produce.**

IMAGE BY VLADITTO

Figure 1. SPDY gateway translates between SPDY-capable clients and conventional HTTP servers.

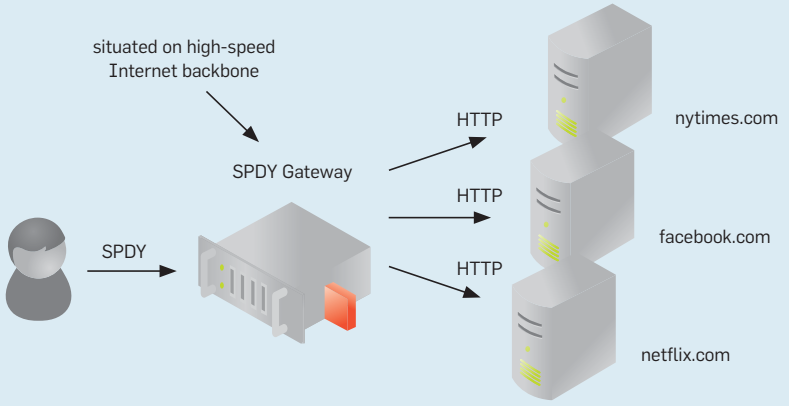
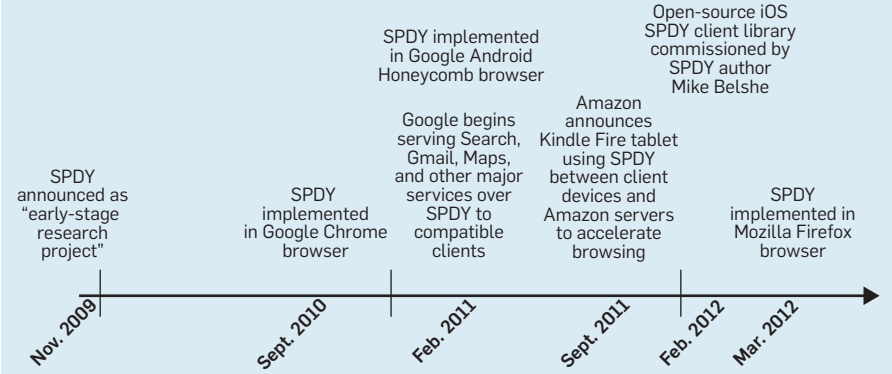


Figure 2. Timeline of SPDY-related development milestones.



November 2009 post[6] on the Google Chromium blog (http://blog.chromium.org) describing the protocol as "an early-stage research project," part of Google's effort to "make the Web faster." By September 2010, SPDY had been implemented in the stable version of the Google Chrome browser,[14] and by February 2011, Google quietly flipped the server-side switch on SPDY, using the protocol to serve major services (such as Gmail, Maps, and search) to Chrome and other SPDY-enabled clients.[4] In February 2011, the Android Honeycomb mobile Web browser received client-side SPDY support, also with little publicity.[13] In September 2011, Amazon announced its Kindle Fire tablet, along with the Silk Web browser that speaks SPDY to Amazon cloud servers.[3] In January 2012, Mike Belshe, SPDY coauthor, commissioned development of an open-source Apple iOS SPDY client library.[20] In March 2012, Firefox 11 implemented the SPDY protocol,[19] which, by June 2012, was enabled by default in Firefox 13,[22] bringing combined client-side support (Chrome + Firefox) to approximately 50% of the desktop browser market[25] (see Figure 3). SPDY is currently an Internet Engineering Task Force (IETF) Internet draft in its third revision.[7]

SPDY's rapid client adoption is impressive, though server adoption lags considerably. SPDY gateways offer a promising transition strategy, bringing many of SPDY's benefits to clients without requiring server support. The incentive for clients to use SPDY gateways is simple: a faster and more secure browsing experience; SPDY is generally deployed over SSL for reasons discussed later. There are also commercial incentives for companies worldwide to deploy SPDY gateways on the high-speed Internet. Content-delivery networks have begun offering SPDY gateway services to Web site owners as a means of accelerating the performance (as experienced by users) of their HTTP Web sites.[1,26] Vendors of mobile devices might deploy SPDY gateways to accelerate the notoriously slow high-latency mobile browsing experience, a marketable feature. Even more intriguing are the incentives for large Web companies to deploy open (publicly available) SPDY gateways to collect and mine rich information about users' Web-browsing behavior, a lucrative commodity in the business of online advertising. Interestingly, the SPDY gateway's ability to aggregate certain critical resources may provide benefits above and beyond regular SPDY, as described later.

SPDY Protocol

SPDY is designed primarily to address performance inhibitors inherent in HTTP, including HTTP's poor support for pipelining and prioritization, the inability to send compressed headers, and lack of resource push capabilities. SPDY's hallmark features—request multiplexing/prioritization, header compression, and server push—are described in the following sections:

Request multiplexing and prioritization. SPDY multiplexes requests and responses over a single TCP connection in independent streams, with request multiplexing inspired by HTTP pipelining while removing several limitations. HTTP pipelining allows multiple HTTP requests to be sent over a TCP connection without waiting for corresponding responses (see Figure 4). Though pipelining has been specified since the 1999 HTTP 1.1 RFC,[11] Opera is the only browser that both implements the feature and enables it by default. Other major browsers either do not implement pipelining or disable pipelining by default, as compatibility with older Web proxies and servers is problematic. Besides a lack of widespread adoption, HTTP pipelining also suffers from head-of-line blocking, as the specification mandates resources be returned in the order they are requested, meaning a large resource, or one associated with a time-consuming

back-end process, delays all other resources (see Figure 5).

SPDY implements pipelining without HTTP's head-of-line blocking limitation. Resources transferred through a SPDY connection are carried in annotated "streams," or independent sequences of bidirectional data divided into frames; annotated streams allow SPDY to not only return resources in any order but interleave resources over a single TCP connection[7] (see Figure 6).

SPDY also includes request prioritization, allowing the client to specify that certain resources be returned with a higher priority than others. Unlike many quality-of-service mechanisms that work on prioritizing packets in queues at lower layers in the network stack, SPDY prioritization works at the application layer, designed to allow the client to specify what is important. One use of prioritization is to request that resources that block progressive page rendering (such as cascading style sheets and JavaScript) be returned with higher priority. Another use of prioritization is to increase the priority of resources being downloaded for the currently visible browser tab while decreasing priority of resources belonging to a currently loading but hidden tab. A further implication of implementing priority at the application layer is that a server can, at least theoretically, prioritize not only the order in which resources are transmitted over the wire but also the order in which resources are generated on the back-end if the task is time intensive.

Header compression. HTTP requests and responses all include a set of HTTP headers that provide additional information about the request or response (see Figure 7). There is significant redundancy in these headers across requests and responses; for example, the "User-Agent" header describing the user's browser (such as Mozilla/5.0 compatible, MSIE 9.0, Windows NT 6.1, WOW64, and Trident/5.0 for Internet Explorer 9) is sent to the server many times over. Likewise, cookie headers, which describe state information about the client, are repeated many times over across requests. Such redundancy means HTTP headers tend to compress relatively effectively. To further improve compression, SPDY seeds an out-of-band compression dictionary based on a priori knowledge of common HTTP headers.[29]

Server push. This advanced feature of SPDY allows the server to initiate resource transfer, rather than having to wait until a corresponding client request is received; Figure 8 outlines a server using it to push a resource to the client that would be requested imminently regardless. Server push saves superfluous round trips by relying on the server's knowledge of resource dependencies to determine what should be sent to the client.

Performance engineers have employed a range of techniques to try to achieve push-like behavior over HTTP, though each involves certain shortcomings; for example, data URIs allow inlining resources (such as images) into the main HTML but are not cacheable and increase resource size by approximately 33%. Another technique, Comet, opens a long-held connection to the server[8] through which arbitrary resources may be pushed but requires an additional initial round trip. Bleeding-edge technologies (such as Web Sockets[28] and resource prefetching[15,21]) also enable push-like behavior but, like Comet, require an additional round trip to establish such behavior. A universal limitation of current techniques is they break resource modu-

Figure 3. Global browser usage share, as recorded by StatCounter (http://gs.statcounter.com/#browser-ww-monthly-201103-201202); in February 2012, Chrome had 29.84% and Firefox 24.88%.

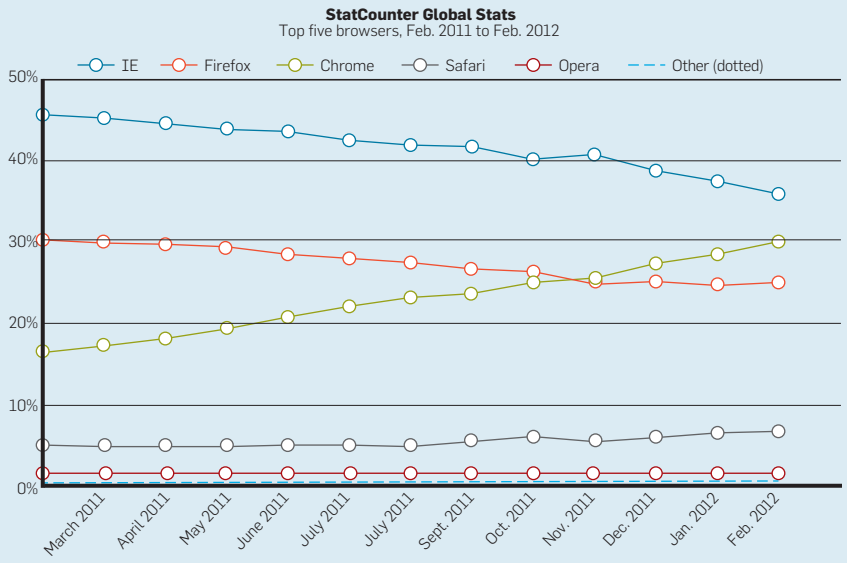
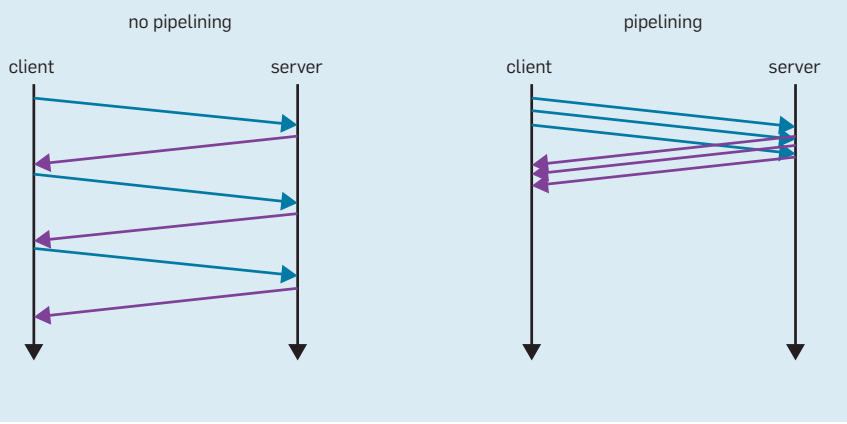


Figure 4. HTTP pipelining allows multiple concurrent requests, reducing the number of round trips to the server.



larity by inserting JavaScript, links, or inlined content into the main HTML document. SPDY push does not require modification of content to support push functionality.

SPDY Security

The SPDY protocol can be run over either a secure (SSL encrypted) or insecure (non-encrypted) transport. In practice, both philosophical views on the role of Web encryption and pragmatism in handling real-world deployment constraints have led to primarily SSL-encrypted implementations. Mike Belshe, SPDY co-author,[7] and Patrick McManus, principal SPDY implementer for Firefox, have expressed their interest in seeing the Web transition to a "secure by default" model.[5,18] Proponents of encrypted SPDY say SSL is no longer computationally expensive[16] and its security benefits outweigh its communication overhead.

The pragmatic reason for deploying SPDY over SSL (port 443) rather than HTTP (port 80) is that transparent HTTP proxies between the client and the server handle HTTP upgrades unreliably.[13] Transparent HTTP proxies do not modify encrypted traffic on SSL port 443 (as they do on port 80) and so should not interfere with newer protocols like SPDY. The rationale for choosing port 443 over an arbitrary port number is that port 443 appears to traverse firewalls more effectively[13] (see Figure 9).

SPDY relies on the next-protocol negotiation (NPN)[17] SSL extension to upgrade the SSL connection to the SPDY protocol. NPN is currently an Internet Engineering Task Force Internet Draft and implemented in OpenSSL[23]; NPN also allows negotiation of other competing future protocols. SSL implementations that do not currently support NPN simply ignore the request to upgrade the protocol, retaining backward compatibility.

Figure 5. Head-of-line blocking in HTTP pipelining; a large resource blocks subsequent smaller resources (left), and a slow-to-generate resource blocks subsequent resources (right).

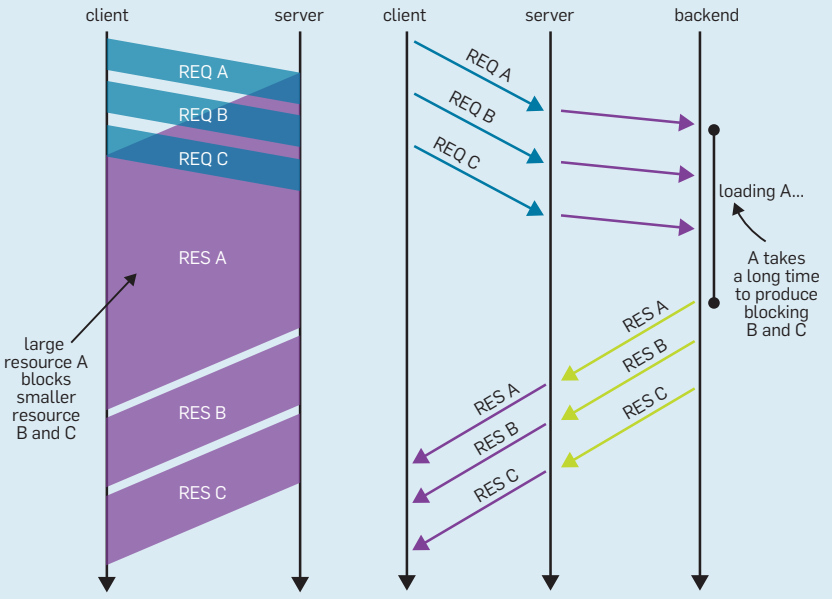
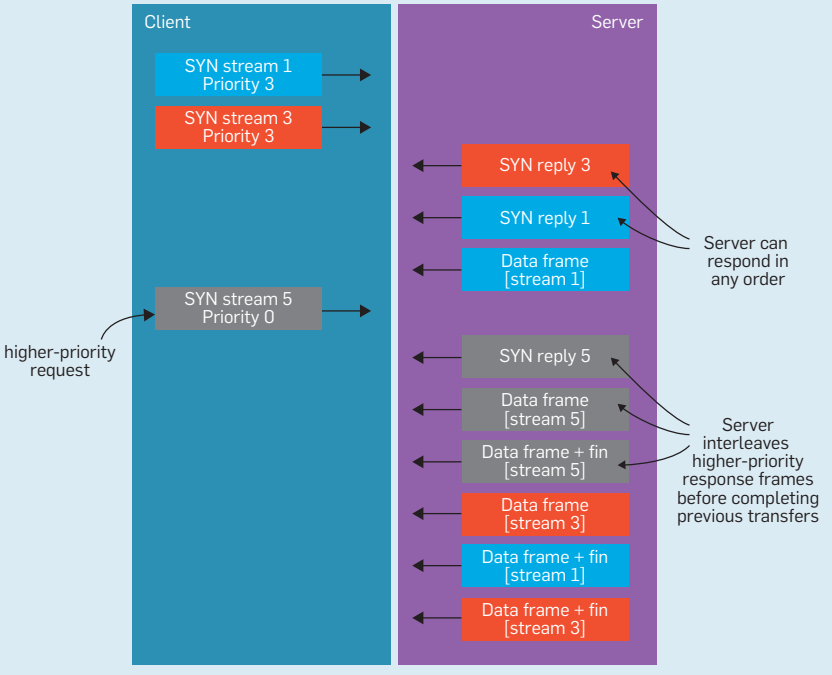


Figure 6. Request multiplexing and prioritization in SPDY; resources are sent in chunks over independent streams that can be interleaved and prioritized.



SPDY Performance

Only a handful of (publicly available) studies quantitatively benchmark SPDY against HTTP, all from the same source—Google. Corroborating the following results across different sites represents important future work:

Google Chrome live A/B study. In 2011, Google benchmarked SPDY against HTTP in real-life A/B tests conducted through the Chrome browser. From March 22 to April 5, 2011, Google configured "in the wild" deployments of the Chrome 12 browser to randomly assign 95% of browser instantiations to use SPDY for SPDY-capable sites; the other 5% of instantiations used HTTPS. Google researchers observed a 15.4% improvement in

page-load time across browser instantiations using SPDY,[13] though a caveat was that domain names in the study were neither recorded nor weighted. Google itself is thought by Google developers to be the current dominant consumer of server-side SPDY technology so is likely overrepresented in these results. Google sites were, in 2011, already heavily optimized, suggesting the stated improvement was likely conservative, though further data is needed for confirmation.

Google's second result from the study was that (encrypted) SPDY is faster than (unencrypted) HTTP for Google's AJAX search; Google researchers provided no further detail.

Google lab tests set one. Google performed a series of laboratory benchmarks of SPDY vs. HTTP under various conditions, though unencrypted SPDY, which would be expected to be faster than encrypted SPDY, was compared against HTTP, despite SPDY deployments being predominantly encrypted.

For simulated downloads of the top 45 pages on the Web (as recorded by Alexa), Google in 2011 reported a 51% reduction in uploaded bytes, 4% reduction in downloaded bytes, and 19% reduction in total packets vs. HTTP.[13] Uploaded bytes were significantly reduced due to SPDY's header compression and the fact that HTTP headers are amenable to strong compression. Google reported that download bytes were only marginally reduced, as most downloaded content in its tests was not headers and in many cases already compressed. The reduction in total packets is due to both a reduction in overall bytes and the fact that SPDY uses only a single connection, resulting in more "full" packets.

Google lab tests set two. A 2009 Google SPDY white paper[14] described simulated page-load time of SPDY vs. HTTP for the Alexa top 25 Web sites. The first test simulated SPDY vs. HTTP with 1% packet loss over simulated home-network connections. Unencrypted SPDY exhibited 27%–60% improvement in page-load time, and encrypted (SSL) SPDY exhibited 39%–55% improvement in page-load time. A second test determined how packet-loss affected SPDY (unencrypted) vs. HTTP; at 0% packet loss, SPDY was 11% faster, and at 2% packet loss SPDY was up to 47% faster. A third test simulated SPDY vs. HTTP as a function of round-trip time; for example, for a 20ms round trip, SPDY was 12% faster than HTTP, and for a 200ms round trip, SPDY was 26% faster; for a full exposition of these results, see Google.[14]

Figure 7. HTTP request and response with example headers; header keys (such as "User-Agent") and header values (such as a particular user agent string) are repeated many times over on a typical connection so make good candidates for compression.

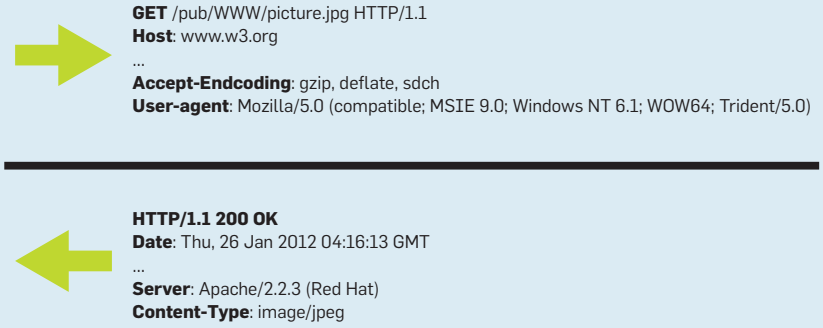
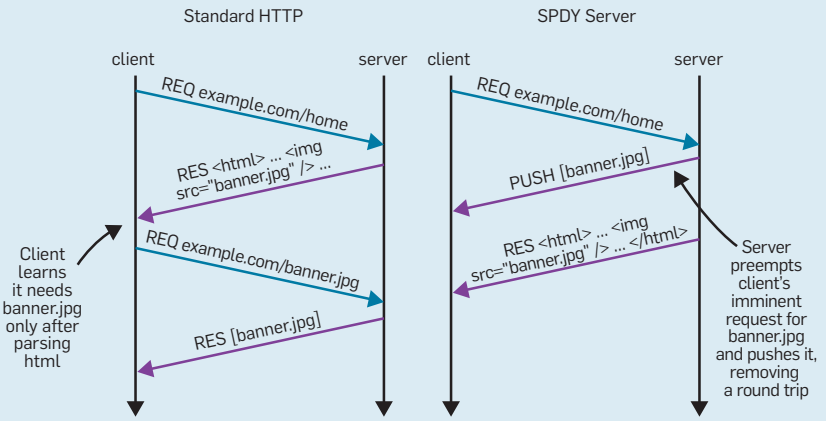


Figure 8. HTTP is unable to push resources to the client, even when it knows the client will require them soon (left); SPDY server push allows the server to initiate the transfer of a resource it believes the client will need soon (right).



Questions on SPDY performance. There is a conspicuous lack of results describing how SPDY performs on mobile devices. SPDY's dominant performance improvements are due in theory to reduced round trips between client and server. Many mobile-carrier technologies today exhibit latency several times that of their fixed-line and Wi-Fi counterparts. By some projections, the majority of the developing world will have its first Internet experience through a mobile carrier, proliferating high-latency connections. In theory, SPDY is ideally suited to these high-latency mobile environments, though real-world results are needed for confirmation.

SPDY push-and-request prioritization is also underexplored. For push, work is needed toward determining how aggressive a server should preemptively push resources to the client. For prioritization, no studies exist on SPDY's efficacy in tabbed browser environments where the currently visible tab's downloading resources could be assigned higher priority.

SPDY Effect on TCP Connections

Though SPDY is an application-layer protocol, it involves broader implica-

tions and interesting interactions with the TCP transport layer:

TCP connection proliferation in HTTP. Prior to standardization of HTTP/1.1 in 1999, HTTP/1.0 permitted downloading only a single resource over a TCP connection and only four concurrent TCP connections to any given server. HTTP/1.1 introduced persistent connections, allowing connection reuse for multiple resources. HTTP/1.1 concomitantly reduced maximum concurrent TCP connections from four to two, helping reduce server load and alleviate Internet congestion[11] induced by proliferation of short-lived TCP connections at the time. Unfortunately, halving concurrent connections had the adverse effect of reducing download parallelism. HTTP/1.1 envisaged that the newly introduced HTTP pipelining would remedy the problem, but, as described earlier, pipelining proved difficult to implement and suffers from head-of-line blocking, as in Figure 5.

Having only two concurrent connections creates a serious performance bottleneck for modern high-speed Internet connections and complex Web sites. First, TCP slow-start, slowly ramping up usable-connection bandwidth based on number of successfully received packets, is often overly conservative in its initial bandwidth allocation. Several round trips are needed before the connection is saturated, by which time much of the content may have been downloaded already (at a slower-than-necessary rate). Second, a typical modern Web page encapsulates 10s or 100s of resources, only two of which may be requested at any given time. Without HTTP pipelining, requests cannot be queued on the server, so each new request incurs an additional round trip. Because most Web resources are small, the round-trip time to the server often dominates over the time to receive the resource from first to last byte.

Modern browsers break from the HTTP/1.1 standard by allowing six or more concurrent TCP connections to a server. This allocation largely circumvents both previously outlined problems—effective initial bandwidth becoming TCP slow-start constant * 6 (rather than * 2) and fewer round trips incurred due to higher request concurrency. A common practice among large Web properties (such as Facebook, Google, and Twitter) is to "shard"[24] a Web page's resources across multiple domains (such as img1.facebook, img2.facebook, img3.facebook, and img4.facebook) to subvert browser policy and obtain greater concurrency. In a modern browser, a Web page sharded across four domains can receive 4 * 6 = 24 concurrent TCP connections.

Figure 9. Real-world success rates of upgrading to newer protocols over various port numbers, as measured by Google Chrome's WebSocket team. Firewalls drop traffic on arbitrary new ports, and transparent proxies inhibit protocol upgrades over port 80.

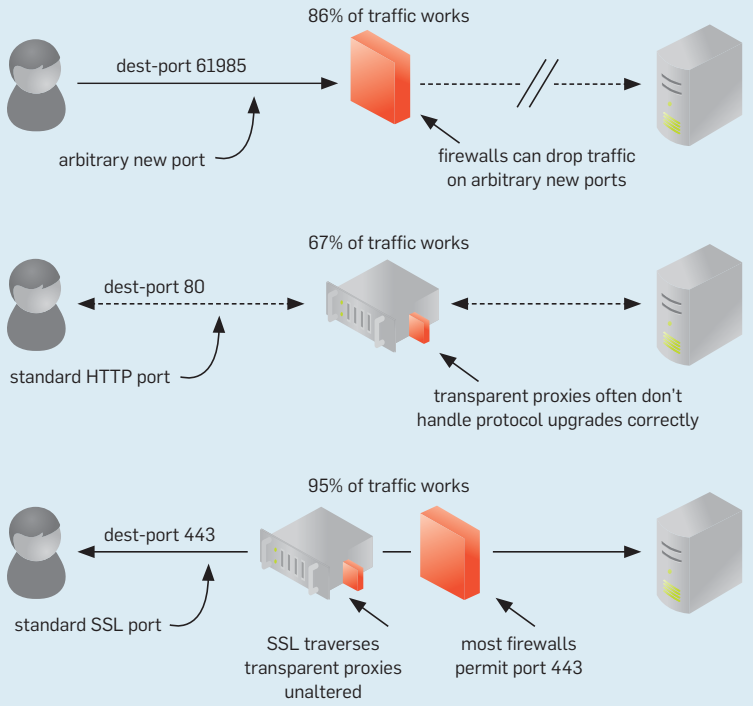
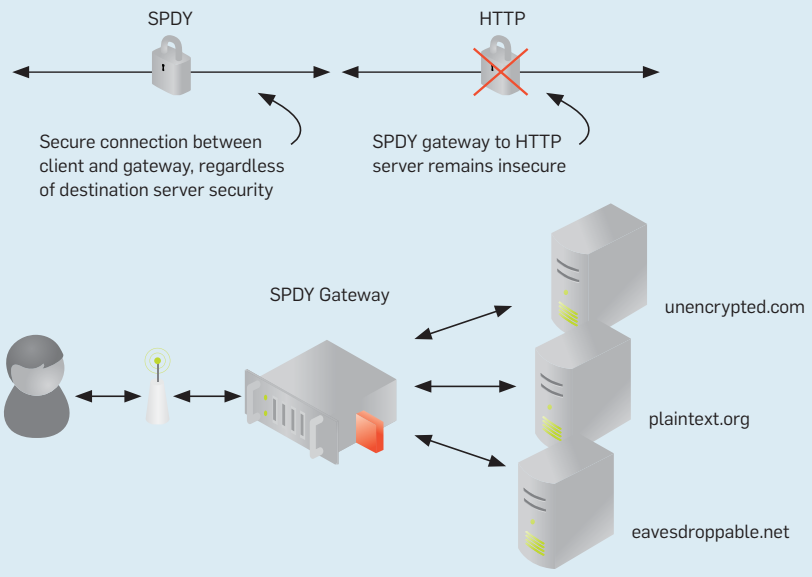


Figure 10. A SPDY gateway offers security between client and gateway regardless of the security of the destination server.



TCP connection proliferation. Increasing concurrent connections through browser policy and sharding can improve page-load time but create other problems. Though highly concurrent connections circumvent an overly conservative slow start on a single TCP connection, they may (in aggregate) exceed total available bandwidth, inducing packet loss. Moreover, the likelihood of losing a critical control packet increases with the number of concurrent connections; for example, the TCP SYN packet, which initiates a TCP connection, has a retransmission timeout measured on the order of seconds if no acknowledgment is received.

Highly concurrent TCP connections also decrease the likelihood of fast retransmit being invoked under packet loss. Fast retransmit is a TCP enhancement that immediately resends a packet without waiting for a fixed timeout delay if acknowledgments for several packets subsequent to the lost packet are received. Highly concurrent connections obtain less bandwidth individually than a single long-lived connection and are therefore less likely to receive and acknowledge enough packets in a short enough duration to trigger fast retransmit. There is also less "body" in each short-lived connection, increasing the likelihood that any packet loss would occur near the end of a connection where too few acknowledgments exist to trigger fast retransmit.

Finally, highly concurrent TCP connections create more connection states to be maintained at various points in the network, including at network-address-translation boxes, as well as state binding processes to TCP port numbers. In some instances, this state can even cause poorly implemented hardware and software to fail or misidentify the highly concurrent connection openings as a SYN flood (a type of denial-of-service attack).[10]

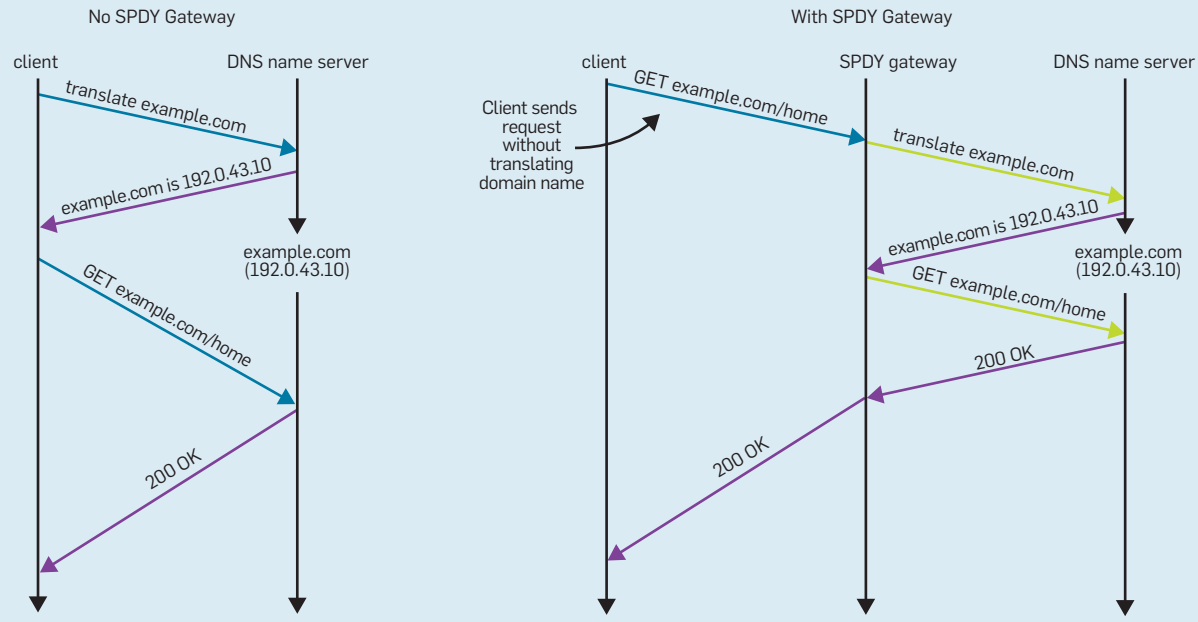
SPDY elephant vs. HTTP mice. The highly concurrent short-lived TCP flows of modern HTTP fall into the category of connections colloquially known as "mice" flows. In contrast, SPDY is able to use a single long-lived "elephant" flow, as it can multiplex and prioritize all requests over a single connection. SPDY therefore retains the benefits of highly concurrent HTTP connections, without detrimental side effects.

A short-term disadvantage of SPDY's single-connection approach is inequitable TCP "backoff" compared to competing applications still using multiple TCP connections; for example, a backoff algorithm that responds to packet loss by reducing the available bandwidth of a connection by 50% will likewise halve the total bandwidth available to an application using a single SPDY TCP connection. The same backoff algorithm applied to an application using 12 concurrent TCP connections would reduce the total available bandwidth to the application by only 4% (1/24) of the connections. Connection proliferation should not be encouraged over the long term, though a short-term mitigation strategy would involve using a small number of concurrent SPDY connections. Long-term research may look to address this issue through smarter backoff algorithms providing equitable treatment to applications, independent of the number of TCP connections.

Transitioning to SPDY

SPDY has been implemented in several popular client browsers, most notably Chrome and Firefox. Though server support for SPDY continues to grow, it has yet to reach the maturity and adoption of client implementations. SPDY gateways are one way to acceler-

Figure 11. A client can delegate DNS lookup to a SPDY gateway, helping minimize round trips.



ate SPDY adoption, providing many SPDY performance and security advantages without requiring SPDY support on the server. A SPDY gateway is an explicit proxy that translates between SPDY-enabled clients and HTTP-only servers. By situating such a gateway on the high-speed Internet, SPDY is used over the slow "last mile" link between the client and the Internet core. The HTTP portion of the connection is in turn isolated to the very-low-latency, very-high-bandwidth link between the gateway and the server, largely mitigating HTTP's dominant performance inhibitors. In addition to providing a practical, viable SPDY transition solution, SPDY gateways also offer several performance-enhancing features:

Secure connection to gateway, regardless of server-side SSL support. Because SPDY operates over SSL, the client-to-gateway connection is secure, regardless of whether SSL is supported on the destination server (see Figure 10). Though the gateway-to-server connection could remain insecure, clients are protected from common attacks (such as eavesdropping on insecure Wi-Fi access points).

Single client-side connection across all domains. As described earlier, SPDY request multiplexing results in dramatically fewer TCP connections than HTTP browsers in use today. However, clients still require at least one new TCP connection for each new server they contact. A SPDY gateway can achieve even greater efficiency than regular SPDY by multiplexing all of a client's requests to the gateway over a single TCP connection covering all servers.

A SPDY gateway might still create multiple connections to a given HTTP server to emulate pipelining and avoid head-of-line blocking but isolate these connections to the high-speed/low-latency Internet core. A SPDY gateway may also retain a small pool of TCP connections to popular servers, allowing new client requests to be forwarded immediately without incurring a new TCP connection handshake or slow-start "warm-up." Likewise, the client needs to perform a single TCP connection handshake only with the gateway and go through the slow-start warm-up only once (as opposed to every time a new server is contacted).

Aside from offering faster browsing as a selling point, Amazon and other potential vendors are likely interested in the data mining and advertising opportunities that come with controlling the gateway.

Delegated DNS lookup. This performance enhancement specific to SPDY gateways entails the gateway performing DNS translations from domain names to server IP addresses on behalf of the client, allowing the client to immediately send a request for a resource to the gateway without knowing the IP address of the server on which it is hosted (see Figure 11). Being situated on the high-speed Internet, the gateway is better positioned to quickly translate the domain name to an IP address; moreover, a gateway that serves a large number of users can cache the IP addresses of popular domains.

Intelligent push. A SPDY gateway can exploit its large user base to infer resource dependencies, even across domains. A regular SPDY-capable server has a limited view of a user's browsing behavior, isolated to the server itself. A gateway sees a user's requests for all servers so it can infer complex patterns of cross-domain navigation; for example, the gateway could determine that 95% of users issuing a Google search query for "Twitter" proceed to twitter.com, and, given this knowledge, the gateway then preemptively pushes resources from the twitter.com homepage to the user. In 2011, Amazon reported the Silk browser on the Kindle Fire tablet already performed intelligent push of this nature, called by Amazon "predictive rendering."[2]

Caching. Like a transparent proxy, a SPDY gateway can cache resources such that subsequent requests for the same resource are served without contacting the origin server.

SPDY gateways, a permanent fixture? This description of SPDY gateways highlights that in some respects gateways offer more attractive features than SPDY directly between clients and servers, including four notable functions: further reduction in TCP connections over the last mile; pre-warming of TCP connections; delegation of DNS translations to the fast Internet core; and intelligent push and resource caching. We suggest that gateways may have a persistent role on the Web, beyond mere transition strategy.

Future SPDY Gateways

Several companies have deployed large SPDY gateways. Perhaps most

notable is the gateway used by the default Silk browser on the Amazon Kindle Fire tablet[2]; Silk proxies much of a user's Web traffic through an Amazon SPDY gateway deployed on the Amazon Web Services cloud infrastructure. Other examples are content-delivery-network/Web-acceleration providers Contendo[1] and Strangeloop,[26] both offering SPDY gateways as a service to HTTP content providers.

Device-specific SPDY gateways. Amazon's decision to couple the Kindle Fire Silk browser to its own proprietary SPDY-based gateway begs the question: Could, and will, other major providers do the same? Could there be, say, an Apple SPDY gateway for iPhones and iPads or a Google SPDY gateway for Android devices in the future? Could such gateways be in the works already? The potential performance advantage of SPDY gateways is particularly intriguing on such resource-constrained mobile devices. The controlled "appliancized" nature of the devices and their operating systems would also simplify vendor implementation. Aside from offering faster browsing as a selling point, Amazon and other potential vendors are likely interested in the data mining and advertising opportunities that come with controlling the gateway.

Open SPDY gateways. Beyond device-specific gateways lies uncharted though potentially lucrative territory—open SPDY gateways—that, like an open proxy, are usable by anyone, independent of device or platform. Major Web companies have demonstrated that free and universal services can be made profitable through related targeted advertising opportunities. So, could SPDY gateways be turned into another free, universal service rendered profitable through better-targeted advertising?

A limitation Web advertisers face today is a restricted view of user activity on domains beyond their direct control. A SPDY gateway provides a vantage point from which to observe all of a user's Web activity, not just on domains under the advertiser's control. Major Web companies like Facebook and Google track users across the Web on third-party sites through partner advertising scripts and other embeddable features (such as the "Like" button), but the picture is incomplete. An open SPDY gateway would provide advertisers missing pieces from the browsing-behavior puzzle that could be fed back into targeted-advertising algorithms. While much the same could be done using device-specific SPDY gateways, an open SPDY gateway would provide insight into a much larger user population. Interesting to consider therefore is whether SPDY gateways (much like search) could become a universal service accessible through a broad range of devices.

Conclusion

SPDY is a high-performance application-layer protocol and potential successor to HTTP. Clients have been quick to adopt it, though server implementations lag. SPDY gateways are helping accelerate SPDY adoption by removing the need for SPDY support on the server. A range of compelling incentives exists for deploying SPDY gateways that are only beginning to be explored. Beyond just a transition strategy, SPDY gateways have performance characteristics that make them attractive for longer-term use. Whether such long-term advantages compared to SPDY support on the server are substantial enough to warrant retaining SPDY gateways is an open question.

Acknowledgments

This work is supported in part by an Australian Government Australian Postgraduate Awards scholarship and Commonwealth Scientific and Industrial Research Organisation Office of the Chief Executive scholarship. The authors would also like to thank the anonymous reviewers for their valuable comments and suggestions.

References

1. Akamai. *Akamai Acquires Contendo.* Press Release, Mar. 2012; http://www.akamai.com/cotendo
2. Amazon. Amazon Silk FAQs; http://www.amazon.com/gp/help/customer/display.html/?nodeId=200775440
3. Amazon. Introducing Amazon Silk; http://amazonsilk.wordpress.com/2011/09/28/introducing-amazon-silk
4. Belshe, M. SPDY on Google servers? Jan. 2011; https://groups.google.com/forum/?fromgroups#!searchin/spdy-dev/SPDY$20on$20Google$20servers?$20/spdy-dev/TCOW7Lw2scQ/INuev2A-ixAJ
5. Belshe, M. SSL: It's a matter of life and death. Mike's Lookout blog, May 28, 2011; http://www.belshe.com/2011/05/28/ssl-its-a-matter-of-life-and-death/
6. Belshe, M. and Peon, R. A 2x faster Web. The Chromium Blog, Nov. 11, 2009; http://blog.chromium.org/2009/11/2x-faster-web.html
7. Belshe, M. and Peon, R. SPDY Protocol. Chromium Projects, Feb. 2012; http://dev.chromium.org/spdy/spdy-protocol/spdy-protocol-draft3
8. Bozdag, E., Mesbah, A., and van Duersen, A. A comparison of push and pull techniques for AJAX in Web site evolution. In *Proceedings of the Ninth IEEE International Workshop* (Paris, Oct. 5–6). IEEE Computer Society, Washington, D.C., 2007, 15–22.
9. Brutlag, J. *Speed Matters for Google Web Search.* Technical Report, 2009; http://services.google.com/fh/files/blogs/google_delayexp.pdf
10. Eddy, W. *TCP SYN Flooding Attacks and Common Mitigations.* Internet Engineering Task Force, Aug. 2007; http://tools.ietf.org/html/rfc4987
11. Fielding, R. et al. *Hypertext Transfer Protocol—HTTP/1.1: Connections.* World Wide Web Consortium, June 1999; http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html
12. Google Inc. mod-spdy: Apache SPDY module. May 2012; http://code.google.com/p/mod-spdy/
13. Google Inc. SPDY essentials. Dec. 2011; http://www.youtube.com/watch?feature=player_detailpage&v=TNBkxA313kk#t=2179s
14. Google Inc. The Chromium Projects. *SPDY: An Experimental Protocol for a Faster Web.* White Paper, 2009; http://www.chromium.org/spdy/spdy-whitepaper
15. Komoroske, A. Prerendering in Chrome. The Chromium Blog, June 2011; http://blog.chromium.org/2011/06/prerendering-in-chrome.html
16. Langley, A. Overclocking SSL. Imperial Violet Blog, June 25, 2010; http://www.imperialviolet.org/2010/06/25/overclocking-ssl.html
17. Langley, A. *Transport Layer Security Next Protocol Negotiation Extension.* Internet Engineering Task Force, Mar. 30, 2011; http://tools.ietf.org/html/draft-agl-tls-nextprotoneg-02
18. McManus, P. Maturing Web transport protocols with SPDY and friends. Video of SPDY Talk at Codebits.eu, Nov. 2011; http://bitsup.blogspot.com.au/2011/11/video-of-spdy-talk-at-codebitseu.html
19. McManus, P. SPDY brings responsive and scalable transport to Firefox 11. Mozilla Hacks blog, Feb. 2012; http://hacks.mozilla.org/2012/02/spdy-brings-responsive-and-scalable-transport-to-firefox-11/
20. Morrison, J. SPDY for iPhone. GitHub, Inc., Jan. 2012; https://github.com/sorced-jim/SPDY-for-iPhone
21. Mozilla Developer Network. Link prefetching FAQ. Mar. 2003; https://developer.mozilla.org/en/Link_prefetching_FAQ
22. Nyman, R. Firefox Aurora 13 is out—SPDY on by default and a list of other improvements. Mar. 19, 2012; http://hacks.mozilla.org/2012/03/firefox-aurora-13-is-out-spdy-on-by-default-and-a-list-of-other-improvements/
23. OpenSSL. OpenSSL Cryptography and SSL/TLS Toolkit. Mar. 2012; http://www.openssl.org/news/changelog.html
24. Souders, S. Sharding dominant domains. Steve Souders blog, May 12, 2009; http://www.stevesouders.com/blog/2009/05/12/sharding-dominant-domains/
25. StatCounter. StatCounter GlobalStats, Feb. 2012; http://gs.statcounter.com/#browser-ww-monthly-201102-201202
26. Strangeloop Networks. Strangeloop. Mar. 2012; http://www.strangeloopnetworks.com/products/overview/
27. The Chromium Projects. SPDY, Mar. 2012; http://dev.chromium.org/spdy
28. World Wide Web Consortium. *The WebSocket API: Editor's Draft 29 August 2012*; http://dev.w3.org/html5/websockets/
29. Yang, F., Amer, P., Leighton, J., and Belshe, M. *A Methodology to Derive SPDY's Initial Dictionary for Zlib Compression.* University of Delaware, Newark, DE, 2012; http://www.eecis.udel.edu/~amer/PEL/poc/pdf/SPDY-Fan.pdf

Bryce Thomas (bryce.m.thomas@gmail.com) is a Ph.D. candidate in the Discipline of Information Technology at James Cook University, Townsville, Queensland, Australia.

Raja Jurdak (raja.jurdak@csiro.au) is a researcher in the Commonwealth Scientific and Industrial Research Organisation and a professor in the University of Queensland, Brisbane, Australia.

Ian Atkinson (ian.atkinson@jcu.edu.au) is a professor and director of the eResearch Centre of James Cook University, Townsville, Queensland, Australia.