Abstract

The design of HTTP—how every application and service on the web communicates today—can positively impact user experience, operational and environmental costs, and even the battery life of the devices you carry around.

Improving HTTP starts with speed. Apps—not just browsers—should get faster too. More and more, apps are how people access web services, in addition to their browser. Improving HTTP should also make mobile better, particularly to ensure great battery life and low network cost on constrained devices. People and their apps should stay in control of network access. Finally, to achieve rapid adoption, HTTP 2.0 needs to retain as much compatibility as possible with the existing Web infrastructure. Done right, HTTP 2.0 can help people connect their devices and applications to the Internet fast, reliably, and securely over a number of diverse networks, with great battery life and low cost.

This document describes "HTTP Speed+Mobility," a proposal for HTTP 2.0 that emphasizes performance improvements and security while at the same time accounting for the important needs of mobile devices and applications. The proposal starts from both the Google SPDY protocol and the work the IETF has done around WebSockets. The proposal is not a final product but rather is intended to form a baseline for working group discussion.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-
Table of Contents

1. Introduction .................................................. 4
   1.1. Overview ................................................... 4
       1.1.1. Layered Architecture ................................. 6
       1.1.2. Existing standards ................................. 6
       1.1.3. Network Cost and Power ............................ 7
       1.1.4. Flexibility in deployment .......................... 8
       1.1.5. Client is in control of content ............. 8
   2. Technical Details .......................................... 9
   3. Acknowledgements .......................................... 10
   4. Normative References ..................................... 11
Authors’ Addresses .............................................. 12
1. Introduction

Over the course of its almost two decades of existence, the HTTP protocol has enabled the web to experience phenomenal growth and change the world in more ways than its creators might have imagined. HTTP's designers got many design principles right, including simplicity and robustness. These characteristics allow billions of devices to support and use HTTP in a multitude of communication scenarios.

Improving HTTP starts with speed. Web sites have become complex. A single site could comprise of hundreds of different elements (from images to videos to ads to news feeds and so on) that need to get retrieved by the client before the page can be fully displayed. Users expect all of this to happen securely and instantly across all their devices and applications. In many scenarios, HTTP fails to meet these expectations. Speed improvements need to apply not only for browsers but also for apps. More and more, apps are how people access web services, in addition to their browser.

The core of the speed problem is that HTTP only allows for a unidirectional request / response model, and it relies on multiple TCP connections for concurrency (pipelining is formally supported by the protocol but is seldom implemented in practice). This leads to a variety of issues, such as additional round trips for connection setup, slow-start delays, and potentially connection rationing: the client may not be able to dedicate too many connections to any single server, and the server needs to protect itself from denial-of-service attacks. As a result, users are often disappointed in the perceived performance of websites.

Improving HTTP should also make mobile better. For example, people want their mobile devices to have better battery life. HTTP 2.0 can help decrease the power consumption of network access. Mobile devices also give people a choice of networks with different costs and bandwidth limits. Embedded sensors and clients face similar issues. Mobile considerations require that HTTP be network efficient while simultaneously being sensitive to the limited power, computation, and connectivity capabilities of the client device.

1.1. Overview

This proposal describes a multiplexing solution to enable efficient delivery of content across a broad variety of scenarios, including mobile apps and devices. It is intended to serve as a baseline for discussion within the HTTPbis working group.

This HTTP proposal adheres to the following principles:
Maintain the existing HTTP semantics. In particular, the meaning of messages traversing a hybrid (1.1/2.0) request chain must be preserved. Any deviation from this principle would represent an extension to HTTP and should be treated as such.

Be as broadly applicable and flexible as the current protocol. As part of that, it must enable servers and clients to select security and compression depending on their own communication needs.

Work with the current Web infrastructure including switches, routers, proxies, load balancers, security systems, DNS servers and NATs.

Account for the needs of modern mobile clients, including power efficiency and connectivity through costed networks.

The proposal’s intended outcome is a protocol that can be quickly and widely adopted in the industry, and start delivering real value to end users without imposing undue burden on hardware and software vendors, as well as administrators of legacy equipment. Implementors should also find it easy to understand due to the familiarity of some of its key concepts, which are aligned with innovations that were adopted in recent HTML5 specifications like WebSockets.

To achieve these goals, this proposal recommends to optimize HTTP without changing its semantics by implementing a session layer between TCP and HTTP that will support multiplexing of multiple HTTP requests/responses. The session layer would have the following properties:

- It would maintain the integrity of the layered architecture.
- It would use an upgrade mechanism similar to that of WebSockets. This would enable compatibility with existing proxies and connection models, without creating a mandatory dependency on TLS.
- [Same as SPDY] The protocol would define two types of frames: data and control.
- [Same as SPDY] The session layer would enable negotiation of multiple simultaneous streams for HTTP requests with minimal overhead.
- [Same as SPDY] The session layer would allow for prioritizing delivery of content to ensure highest value traffic is delivered first.
The session layer would enable future extensions to HTTP 2.0 for new scenarios like server push (those extensions would be documented separately).

These properties are described in more detail below.

1.1.1. Layered Architecture

HTTP relies on an in-order, reliable transport to ensure delivery of application data. TCP has almost exclusively provided the reliable, ordered delivery of HTTP messages from one computer to another since its inception. TCP accounts for adverse network conditions such as congestion, or other unpredictable network behavior. Any HTTP 2.0 proposal should leverage the reliable transport and not attempt to replicate functions generally accepted as addressed by other layers.

Conversely, any proposals for enhancing functionality typically provided by other layers of the networking stack (e.g. congestion control provided by the transport layer) should be brought to the attention of, and discussed in, proper IETF forums (e.g. TCPM WG).

During the charter proposal discussion, the security and applications area directors suggested an additional paragraph on security work and authentication. If new work is undertaken in this regard, it should be done by existing IETF security groups in this area.

1.1.2. Existing standards

HTTP at its core is a simple request-response protocol. The working group has clearly stated that it is a goal to preserve the semantics of HTTP. Thus, we believe that the request-response nature of the HTTP protocol must be preserved. The core HTTP 2.0 protocol should focus on optimizing these HTTP semantics, while improving the transport via a new session layer. Additional capabilities that introduce new communication models like unrequested responses must be treated as an extension to the core protocol, and explored separately from the core protocol.

Additionally, HTTP 2.0 should prefer models that are compatible with the existing Internet and, where possible, reuse existing protocol mechanisms. One primary example is in protocol negotiation where the WG should avoid a proliferation of methods, and instead consider using the HTTP 1.1 Upgrade header as it is used in the WebSocket protocol. This will help HTTP 2.0 to be readily deployed on the existing internet, and maintain compatibility with existing web sites and client environments (such as some educational networks).
1.1.3. Network Cost and Power

Any new protocol for transporting HTTP data on the Internet must also take into account the types of systems and devices that use HTTP and how they are connected to the Internet. The growth of the Internet of the next decade (and longer) will be fueled by mobile apps and mobile devices, as well as by the cheap, limited-capability devices envisioned by the "Internet of Things." For all these devices, speed is only one design tenet: considerations about battery life, bandwidth limitations, processor and memory constraints, and various policy mandates will also challenge designers and users. For instance, the user of a device connected over mobile broadband may need to minimize the amount of data sent in order to conserve bandwidth, minimize power usage and monetary cost of communication. Furthermore, transmitting the same amount of data may have radically different power implications depending on how the transfer is structured: for example, when operating over a mobile broadband interface it is more efficient to use a single larger transfer than to space out the transmission in multiple smaller transfers. Multiple transfers may cause multiple radio transitions between low and high powered states, causing additional battery drain.

In short, the choice among speed, cost, and power is not a simple one. At times, speed may be the most important consideration. Other times, bandwidth cost or battery life may be the deciding factor. HTTP 2.0 must allow developers to optimize for the specific constraints of their problem space (which might change over time) rather than imposing a "one size fits all" solution to a generic problem. For example, a server push extension could be a good optimization for many scenarios where content updates to web pages revisited over time are infrequent, and where the client has plenty of bandwidth as well as the needed processing power to either handle the updates instantly, or cache them for later processing. On the other hand, it is not likely to be appropriate in situations where content is being transmitted over a costed link. Neither it will be when the client is running several applications that use network bandwidth concurrently, and bursty, server-initiated content transmissions would interfere with their smooth operation. Rather than forcing developers to choose between using all the features of HTTP 2.0 or sticking with HTTP 1.1, it would be better to provide mechanisms for developers to fine tune the capabilities of HTTP 2.0 to a specific set of requirements.

In summary, the goals of higher speed, lower cost, lower power may often be aligned. For instance, having less data sent on the wire will allow pages to load faster, allow the radio to power down sooner and consume less bandwidth. But given the variety of the scenarios where HTTP 2.0 will be used, this will not always be the case. For
example, a device whose battery is about to run out, or whose cache is near capacity can provide a better user experience by disabling all (or most) server push updates while retaining the other optimizations available in HTTP 2.0. Accordingly, the working group should consider power and cost as well as speed in designing extensions to HTTP 2.0.

1.1.4. Flexibility in deployment

HTTP is used in a vast array of scenarios and a variety of network architectures. There is no "one size fits all" deployment of HTTP. For example, at times it may not be optimal to use compression in certain environments. For constrained sensors from the "Internet of things" scenario, CPU resources may be at a premium. Having a high performance but flexible HTTP 2.0 solution will enable interoperability for a wider variety of scenarios. There also may be aspects of security that are not appropriate for all implementations. Encryption must be optional to allow HTTP 2.0 to meet certain scenarios and regulations. HTTP 2.0 is a universal replacement for HTTP 1.X, and there are some instances in which imposing TLS is not required (or allowed). For example, a "random thought of the day" web service has very little need for it, nor does a sensor spewing out a temperature reading every few seconds.

1.1.5. Client is in control of content

Because of the variety of clients on the Internet and the number of connection scenarios, clients must be able to define what content is downloaded. The app or browser is in the best position to assess what the user is currently doing and what data is already locally available. For example, most of the browsers in use today have powerful caches that should be leveraged to store web elements that change infrequently. Clients need the ability to inform the server about cached elements that do not need to be downloaded. In addition, a particular client may have security and compatibility needs with regard to the data being sent. HTTP 2.0 proposals should not force the client to download content that has not been requested and may already be cached. Furthermore, the client must have the option to decline unwanted or unneeded content. Ideally this feedback from the client to the server would allow for incremental approval of content to enable an efficient "push" extension to deliver the right content.
2. Technical Details

To be added within the next few days in version 01.
3. Acknowledgements

Thanks to the following individuals who have also contributed with discussions and text: Brian Raymor, Ravi Rao, Dave Thaler, Ivan Pashov, Jitu Padhye, Jean Paoli, Michael Champion, NK Srinivas, Sharad Agarwal and Rob Mauceri.

4. Normative References

