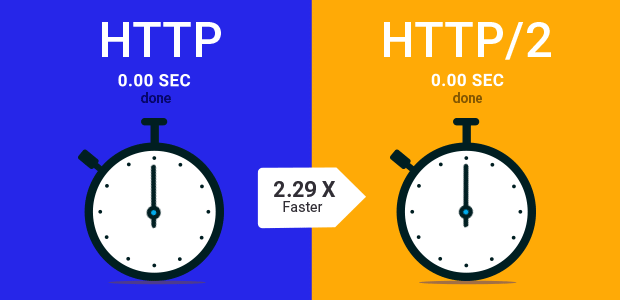
**1- پروتکل HTTP**

**سوال 1**

پروتکل جدید HTTP/2 باعث می‌شود صفحات وب بطور قابل توجهی با سرعت بالاتری در دسکتاپ و تلفن‌های همراه بارگزاری شوند

.

**تفاوت‌های کلیدی HTTP2 و HTTP1.x:**

**-1 نسخهٔ HTTP/2 حاوی داده‌های باینری (دودویی) است:**HTTP/1.1 از داده‌های متنی استفاده می‌کند و این در حالی است که داده‌های متنی به طور کلی در سراسر شبکه از بازدهی کمتری نسبت به داده‌های باینری برخوردارند.

**-2هِدِرهای HTTP/2 فشرده‌ شده هستند:**به طور کلی منظور از Header اطلاعاتی است که در پاسخ به یک ریکوئست ارسال می‌شود که شامل دیتا، مبدأ، نوع، حجم، مدت زمان کَش و موارد دیگر است. برخلاف HTTP/1.1، این داده‌ها در نسخهٔ HTTP/2 فشرده‌سازی می‌شوند تا پرفورمنس ارتقاء یابد.

**-3نسخهٔ HTTP/2 اصطلاحاً Asynchronous است:**در HTTP/1.1، سرور باید به همان ترتیبی که ریکوئست‌ها را دریافت کرده است، ریسپانس‌ها را ارسال کند اما نسخهٔ HTTP/2 اصطلاحاً Asynchronous است؛ بنابراین پاسخ‌های سریع‌تر و در عین حال با حجم کمتری می‌تواند در زمان کوتاه‌تری از سمت سرور ارسال شود.

**-4 نسخهٔ HTTP/2 مولتی‌پلکس است:**در HTTP/1.1، فقط یک درخواست روی یک کانکشن اینترنتی TCP در آن واحد می‌تواند به کار گرفته شود و مرورگرها به طور عادی قادر به ایجاد 4 تا 8 کانکشن با سرور هستند و این در حالی است که ریکوئست‌هایی با حجم زیاد می‌توانند سرعت دانلود فایل‌های دیگر را به تأخیر بیندازند! HTTP/2 اجازهٔ ارسال چندین ریکوئست (درخواست) و دریافت ریسپانس (پاسخ) از سمت سرور را به طور هم‌زمان بر روی یک کانکشن امکان‌پذیر می‌سازد.

**-5 نسخهٔ HTTP/2 امکان استفاده از Server Push را فراهم می‌سازد: ‌‌**با استفاده از این نسخه از پروتکل اچ‌تی‌تی‌پی، سرور می‌تواند فایل‌ها -و به طور کلی هر نوع داده‌ای- را قبل از آنکه ریکوئستی ارسال شود، برای مرورگر بفرستد که به این فناوری اصطلاحاً **Server Push** گفته می‌شود. برای مثال، ممکن است شما در پایین صفحهٔ خود به یک اسکریپت لینک دهید. در HTTP/1.1، مرورگر کدهای HTML را دانلود می‌کند، تجزیه می‌کند و سپس فایل جاوااسکریپت را بارگذاری می‌کند (این بارگذاری هنگامی است که با تگ <script> روبه‌رو شویم). سروری که HTTP/2 را ساپورت کند، می‌تواند چنین فایلی را قبل از اینکه نیاز آن را تشخیص دهد، برای مرورگر ارسال کند که در نتیجه در صورت نیاز، کاربر معطل دانلود شدن فایل‌های جی‌اس نخواهد شد (که این به معنی UX بهتر است).

**سوال 2**

HTTP/3 is designed for QUIC, which is a transport protocol that handles streams by itself.

HTTP/2 is designed for TCP, and therefore handles streams in the HTTP layer.

# Similarities

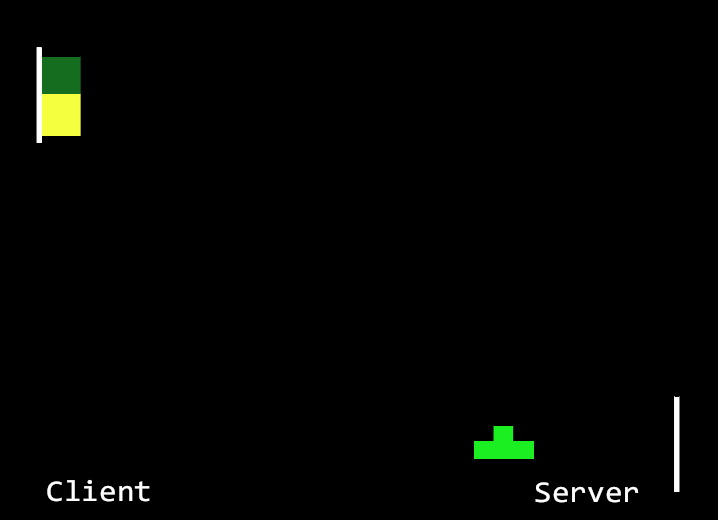
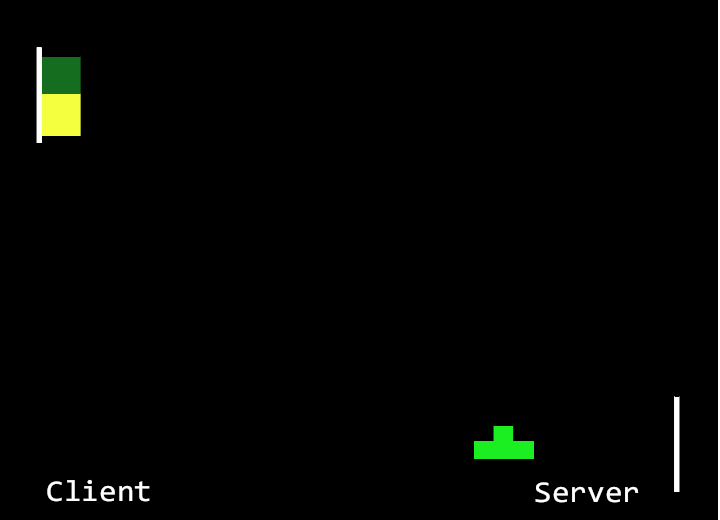
The two protocols offer clients virtually identical feature sets.

* Both protocols offer server push support
* Both protocols have header compression, and QPACK and HPACK are similar in design.
* Both protocols offer multiplexing over a single connection using streams

# Differences

The differences are in the details and primarily there thanks to HTTP/3's use of QUIC:

* HTTP/3 has better and more likely to work early data support thanks to QUIC's 0-RTT handshakes, while TCP Fast Open and TLS usually sends less data and often faces problems.
* HTTP/3 has much faster handshakes thanks to QUIC vs TCP + TLS.
* HTTP/3 does not exist in an insecure or unencrypted version. HTTP/2 can be implemented and used without HTTPS - even if this is rare on the Internet.
* HTTP/2 can be negotiated directly in a TLS handshake with the ALPN extension, while HTTP/3 is over QUIC so it needs an Alt-Svc: header response first to inform the client about this fact.
* HTTP/3 has no prioritization. The HTTP/2 approach to prioritization has been deemed too complicated, or even a downright failure, and there's work on creating a simpler take. This planned simpler scheme is also planned to be able so backport to run over HTTP/2 using HTTP/2's extension mechanism.



The above shows HTTP/3 multiplexing 2 requests. A packet is lost that affects the yellow response but the green one proceeds just fine.

**سوال 3**

HTTP 3xx status codes indicate that further action is needed. The request was received, but can’t be fulfilled yet. When the client receives a 3xx status code, it needs to make a new request to the location the server returned along with the redirect. The HTTP 3xx status codes are all redirects.

**سوال 4**

**Host**

The **Host** **request** **header** specifies the **host** and **port** number of the server to which the request is being sent.

**Referer**

The **Referer** **request header** contains **the address of the page making the request**. When following a link, this would be the url of the page containing the link. When making AJAX requests to another domain, this would be your page's url. The Referer header allows servers to identify **where people are visiting them from** and may use that data for **analytics**, **logging**, or **optimized** **caching**, for example.

**Accept-Encoding**

The **Accept-Encoding** **request HTTP header** advertises **which content encoding**, usually a compression algorithm**, the client is able to understand**. Using [content negotiation](https://developer.mozilla.org/en-US/docs/Web/HTTP/Content_negotiation), the server selects one of the proposals, uses it and informs the client of its choice with the [Content-Encoding](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Encoding) response header.

**Content-type**

**In responses**, a Content-Type **header** tells the client what the **content type of the returned content** actually is. Browsers will do MIME sniffing in some cases and will not necessarily follow the value of this header; to prevent this behavior, the header [X-Content-Type-Options](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options) can be set to nosniff.

**Content-Length**

The **Content-Length** **entity header** **(request & response)** indicates **the size of the entity-body**, in bytes, sent to the recipient.

**Content-Range**

The **Content-Range** **response HTTP header** indicates **where** in a full body message **a partial message belongs**.

**Location**

The **Location** **response header** indicates **the URL to redirect a page to**. It only provides a meaning when served with a 3xx (redirection) or 201 (created) status response.

**Last-Modified**

The **Last-Modified** **response HTTP header** contains the **date and time** at which the origin server believes the **resource was last modified**. It is used as a **validator** to determine **if a resource** received or stored **is the same**. Less accurate than an [ETag](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag) header, it is a fallback mechanism. Conditional requests containing [If-Modified-Since](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/If-Modified-Since) or [If-Unmodified-Since](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/If-Unmodified-Since) headers make use of this field.

**Cache-Control**

The **Cache-Control** HTTP header holds directives **(instructions) for**[**caching**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching) **in** **both** **requests** **and** **responses**. A given directive in a request does not mean the same directive should be in the response.