Click to prove
you're human

Find free, ready-to-use math resources for algebra, geometry, number sense, measurement, operations, statistics and probability across grades 4-8 and high school to enhance student exploration and practice!Check out some fun and interactive examples handpicked by our staffExplore the wide range of resources created by the GeoGebra Content Team to support your students' learning needsExplore an extensive collection of over one million math and science activities, exercises and lessons meticulously crafted by our global GeoGebra community. Immerse yourself in the boundless possibilities that await you.Create a free account so you can save your progress any time and access thousands of math resources for you to customize and share with others Share — copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt — remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution — You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. Free mathematics software with utilities for performing algebra, calculus and geometry which creates different visual flat and 3D graphs.GeoGebra, an open-source utility designed explicitly for helping users design diverse mathematical objects used for calculus, algebra and geometry. Since it is a Java-based application, ensure that the working environment has been installed previously on the computer.Lovely UIWith a user-friendly interface and clear-cut options, it offers quick access to different drawing tools that can be easily inserted in the working pane. GeoGebra empowers you to add points and vectors, perform calculations with them, and include lines and axes, conic sections and arcs, curves, inequalities, intervals, and functions.Take advantage of a wide array of math functions, including number creation, C0 and Euler constants for expressions and calculations, and work with angles, Boolean variables, matrix operations, embedded text messages, and images.Customizable visualizationsCustomize object properties by revealing or hiding the inserted objects in graphical representation, filling them with a color, fixing them to a certain position so that they cannot be moved, redefined or deleted. You can also change the name of an object, label it with its value, create animations with numbers, angles, or points, and enable the tracing mode for viewing an object’s position.Scripting and exportingGeoGebra offers support for two scripting languages (GGBScript and Javascript), allowing you to create a sequence of commands for designing or modifying objects. Additionally, you can record the values for each object in a spreadsheet list, including numbers, points, and vectors.The projects you generate can be saved to a file (e.g. HTML, PNG, EPS, GIF), uploaded on the developer’s website, or printed.To sum it upIn conclusion, GeoGebra comes packed with many dedicated parameters that help you learn or teach various mathematical objects and operations.Features of GeoGebra3D Graphics: Generate 3D models and rotate and zoom them.Animations: Visualize changes of graphs with time and dynamic sliders.CAS: Access an integrated Computer Algebra System for exact calculations.Calculations: Perform algebraic computations and number crunching.Exports: Save your work as images, videos, web pages or other formats.Geometry: Construct interactive geometric figures and measure properties.Graphs: Create 2D and 3D plots and view them from different perspectives.Interactivity: Add interactive elements such as buttons, checkboxes and input fields.Measurement: Measure lengths and angles accurately in 3D figures.Polynomials: Factorize and expand polynomials with the Polynomial Tool.Primitives: Use basic primitives such as points, lines, circles and polygons.Scripting: Create custom scripts to automate and extend GeoGebra.Spreadsheet: Organize your data in tables and create charts.Spreadsheet Functions: Use spreadsheet functions to analyze data.Statistics: Analyze data and generate statistical diagrams.Compatibility and LicenseGeoGebra is provided under a freeware license on Windows from calculators with no restrictions on usage. Download and installation of this PC software is free and 6.0.887.0 is the latest version last time we checked.What version of Windows can GeoGebra run on?GeoGebra can be used on a computer running Windows 11 or Windows 10. Previous versions of the OS shouldn't be a problem with Windows 8 and Windows 7 having been tested. It comes in both 32-bit and 64-bit downloads.Other operating systems: The latest GeoGebra version from 2025 is also available for Mac.Filed under: GeoGebra DownloadFree CalculatorsMath Calculation Software This repository contains source code of GeoGebra's math apps. It is available on a private GitLab instance and mirrored to GitHub. Please read about GeoGebra's licensing. To start the web version from command line, run This will start a development server on your machine where you can test the app. If you need to access the server from other devices, you can specify a binding address ./gradlew :web:run -Pgbind=A.B.C.D where A.B.C.D is your IP address. Then you can access the dev server through . You can also run ./gradlew :web:tasks to list other options. To start the desktop version from command line, run You can also run ./gradlew :desktop:tasks to list other options. Open IntelliJ. If you don't have IntelliJ on your computer yet then you can download and install it from here In the menu select File / New / Project from Version Control / Git In the new window add the following path: Click on ‘Checkout’, select your preferred destination folder, select Java 1.8 as the SDK, click on OK and wait... After the project is checked out, select the root folder of the project, open the Run Anything tool (Double ^ on Mac) and run the following command: ./gradlew :web:run After a minute or two the GWT UI will appear After the Startup URLs are loaded on the UI, select the app that you wish start. For example, if you select graphing.html and click on Launch Default Browser then the Graphing Calculator app with the newest features will load and start in your default browser Create a free account so you can save your progress any time and access thousands of math resources for you to customize and share with others GeoGeobra is an intelligent graphing software that allows the user to interactively explore 2D and 3D Cartesian & Euclidean geometry - as well as calculus. Best of all - it's a free offering! You can download it from . From their blurb: GeoGebra is a free and multi-platform dynamic mathematics software for schools that joins geometry, algebra and calculus. Building an Interactive Document in GeoGebra Let's go through the process of creating a document in GeoGebra. Our document will allow the end-user to explore the changing slope of a polynomial as x changes value. First, we enter the function at the bottom of the screen, using the carat "^" for powers of x: The program converts the function display (see under "Free objects") so that it is more easily read by a human. I have scaled the y-axis by clicking on the "Move" tool (the one on the far top right) and simply dragging the axis to the desired scale. Next, we are going to add a tangent line to our curve. We add a new point on the function using the "New point" tool: We place the new point anywhere on the curve by clicking on the curve. It will now "stick" to the curve as we drag it. Note the other tools that are available on this tool item: Intersection of 2 two curves Midpoint between 2 points Now we choose the "Tangent" tool as follows: Note the other tools that are available on this drop-down. You can construct: Perpendicular line through a point Parallel line through a point Perpendicular bisector Bisector of an angle Diameter line of a conic section Helpful hints appear on the GeoGebra interface that tell us to click on the point, then the curve. We now have a tangent line. The exploratory activity we can do now is to drag the point "A" to any position on the curve (after selecting the "pointer" icon at the far left) and the tangent line follows along. Even better, we can get a readout of the actual slope as we move around the curve, by typing in: s = Slope[a] Rather than just giving a numerical value for the slope, it actually gives a triangle with base length 1 unit, indicating more clearly what a slope at a point really means. The result for one part of the curve is as follows: Let's now add the curve of the first derivative to our existing plot. (Of course, we expect the first derivative curve to be a parabola, since it will be a polynomial of degree 2). We achieve this by entering: Derivative[f] The green curve is the first derivative curve (a parabola, as expected): We can trace the locus of a point (B) moving on the first derivative curve, as follows: To create the point B, I entered: B = (x(A), f'(x(A))) This means the point B will have the same x-value as A (we write this using x(A)), and the y-value will be the same as the y-value of the first derivative curve, which I wrote with f'(x(A)). You can use GeoGebra to examine critical points like local maximums and minimums on the curve and the point of inflection (point A) illustrated above. Other Tools GeoGebra is a feature-rich offering. The other tools available in GeoGebra that I have not already mentioned include: Rotate an object around a point Draw line segments Draw vectors Draw polygons (including regular polygons) Construct various circles, arcs and sectors Angles, distances and areas You can add text and images You can zoom in and out on objects Split Functions GeoGebra will draw piece-wise functions (with a little coaxing). You can achieve the following (with a vector thrown in): Output I like the variety of output options. You can either: Save your file for later use (it will have a .GGB extension) Save the graph as an image in PNG, EPS, SVG or EMF format Save the graph to the clipboard (for manipulation in an image editing program or for pasting into a document) Save as an interactive Web page, but this can only be uploaded to GeoGebra Tube (not to your hard drive) Euclidean Geometry GeoGebra allows you to easily create angles, polygons and conics. As you can see in the regular dodecagon above, GeoGebra allows you to measure angles, including internal angles. Output and mobile apps There is now a HTML5 export option in GeoGebra (so it can be run on Web pages without java) and there are a range of apps for iOS, Android, Windows, Mac, Chromebook and Linux. 3D Graphs Geogebra 5 has 3D graph capability. It has been quite a wait, but it's been worth it. Here's a graph of a(x,y) = x^2 + cos(y) You can easily zoom in and out, and rotate the graph left-right and up-down to see what it looks like from all angles. In 2D graphs, where you can re-scale the graph by dragging either the x- or y-axis. However, this feature is missing from 3D graphs (so far, at least). Resources There is a vibrant GeoGebra community that provides a lot of free help and resources. GeoGebraTube has thousands of ready-made interactive files made with GeoGebra. The GeoGebra Forum has many experts who are willing to help with any issues you face. Conclusion GeoGebra is an impressive geometry and calculus exploratory tool. I tend to use it as an exploratory tool when I need something quick. It's a great tool, but I tend to stick to JSXGraph when developing interactive graphs of IntMath. GeoGebra is more intelligent than MS Math 4.0, which I reviewed earlier (it has 3D capability - see Microsoft Math 4.0), but the audience for each product is not exactly the same. Having the 2 products will give you some excellent tools for exploring mathematics. Do yourself a favor - download GeoGebra now! See the 4 Comments below. Application that uses a web browser as a client This article has multiple issues. Please help improve it or discuss these issues on the talk page. (Learn how and when to remove these messages) This article needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.Find sources: "Web application" – news · newspapers · books · scholar · JSTOR (February 2018) (Learn how and when to remove this message) This article possibly contains original research. Please improve it by verifying the claims made and adding inline citations. Statements consisting only of original research should be removed. (May 2022) (Learn how and when to remove this message) This article may need to be rewritten to comply with Wikipedia's quality standards. You can help. The talk page may contain suggestions. (May 2022) (Learn how and when to remove this message) Screenshot from 2007 of Horde, a groupware and open-source web application A web application (or web app) is application software that is created with web technologies and runs via a web browser.[1][2] Web applications emerged during the late 1990s and allowed for the server to dynamically build a response to the request, in contrast to static web pages.[3] Web applications are commonly distributed via a web server. There are several different tier systems that web applications use to communicate between the web browsers, the client interface, and server data. Each system has its own uses as they function in different ways. However, there are many security risks that developers must be aware of during development; proper measures to protect user data are vital. Web applications are often constructed with the use of a web application framework. Single-page applications (SPAs) and progressive web apps (PWAs) are two architectural approaches to creating web applications that provide a user experience similar to native apps, including features such as smooth navigation, offline support, and faster interactions. The concept of a "web application" was first introduced in the Java language in the Servlet Specification version 2.2, which was released in 1999. At that time, both JavaScript and XML had already been developed, but the XMLHttpRequest object had only been recently introduced on Internet Explorer 5 as an ActiveX object.[citation needed] Beginning around the early 2000s, applications such as "Myspace (2003), Gmail (2004), Digg (2004), [and] Google Maps (2005)," started to make their client sides more and more interactive. A web page script is able to contact the server for storing/retrieving data without downloading an entire web page. The practice became known as Ajax in 2005. In earlier computing models like client-server, the processing load for the application was shared between code on the server and code installed on each client locally. In other words, an application had its own pre-compiled client program which served as its user interface and had to be separately installed on each user's personal computer. An upgrade to the server-side code of the application would typically also require an upgrade to the client-side code installed on each user workstation, adding to the support cost and decreasing productivity. Additionally, both the client and server components of the application were bound tightly to a particular computer architecture and operating system, which made porting them to other systems prohibitively expensive for all but the largest applications. Later, in 1995, Netscape introduced the client-side scripting language called JavaScript, which allowed programmers to add dynamic elements to the user interface that ran on the client side. Essentially, instead of sending data to the server in order to generate an entire web page, the embedded scripts of the downloaded page can perform various tasks such as input validation or showing/hiding parts of the page. "Progressive web apps", the term coined by designer Frances Berriman and Google Chrome engineer Alex Russell in 2015, refers to apps taking advantage of new features supported by modern browsers, which initially run inside a web browser tab but later can run completely offline and can be launched without entering the app URL in the browser. This section relies excessively on references to primary sources. Please improve this section by adding secondary or tertiary sources. Find sources: "Web application" – news · newspapers · books · scholar · JSTOR (November 2022) (Learn how and when to remove this message) Traditional PC applications are typically single-tiered, residing solely on the client machine. In contrast, web applications inherently facilitate a multi-tiered architecture. Though many variations are possible, the most common structure is the three-tiered application. In its most common form, the three tiers are called presentation, application and storage. The first tier, presentation, refers to a web browser itself. The second tier refers to any engine using dynamic web content technology (such as ASP, CGI, ColdFusion, Dart, JSP/Java, Node.js, PHP, Python or Ruby on Rails). The third tier refers to a database that stores data and determines the structure of a user interface. Essentially, when using the three-tiered system, the web browser sends requests to the engine, which then services them by making queries and updates against the database and generates a user interface. The 3-tier solution may fall short when dealing with more complex applications, and may need to be replaced with the n-tiered approach; the greatest benefit of which is how business logic (which resides on the application tier) is broken down into a more fine-grained model.[4] Another benefit would be to add an integration tier, which separates the data tier and provides an easy-to-use interface to access the data.[4] For example, the client data would be accessed by calling a "list_clients()" function instead of making an SQL query directly against the client table on the database. This allows the underlying database to be replaced without making any change to the other tiers.[4] There are some who view a web application as a two-tier architecture. This can be a "smart" client that performs all the work and queries a "dumb" server, or a "dumb" client that relies on a "smart" server.[4] The client would handle the presentation tier, the server would have the database (storage tier), and the business logic (application tier) would be on one of them or on both.[4] While this increases the scalability of the applications and separates the display and the database, it still does not allow for true specialization of layers, so most applications will outgrow this model.[4] Main article: Internet security This section needs additional citations for verification. Please help improve this article by adding citations to reliable sources in this section. Unsourced material may be challenged and removed. (February 2018) (Learn how and when to remove this message) Security breaches on these kinds of applications are a major concern because it can involve both enterprise information and private customer data. Protecting these assets is an important part of any web application, and there are some key operational areas that must be included in the development process.[5] This includes processes for authentication, authorization, asset handling, input, and logging and auditing. Building security into the applications from the beginning is sometimes more effective and less disruptive in the long run. Writing web applications is simplified with the use of web application frameworks. These frameworks facilitate rapid application development by allowing a development team to focus on the parts of their application which are unique to their goals without having to resolve common development issues such as user management.[6] In addition, there is potential for the development of applications on Internet operating systems, although currently there are not many viable platforms that fit this model.[citation needed] Internet portal Web API Software as a service (SaaS) Web 2.0 Web engineering Web GIS Web services Web sciences Web widget ^ "Web app | Definition, History, Development, Examples, Uses, & Facts | Britannica". www.britannica.com. Encyclopædia Britannica. Retrieved 4 November 2024. ^ "What is a Web App? - Web Application Explained - AWS". Amazon Web Services, Inc. Retrieved 4 November 2024. ^ "Web applications". DocForge. Archived from the original on 19 April 2015. Retrieved 9 November 2024. ^ a b c d e f Petersen, Jeremy (4 September 2008). "Benefits of using the n-tiered approach for web applications". Archived from the original on 1 December 2017. Retrieved 24 November 2017. ^ "Top Tips for Secure App Development". Dell.com. Archived from the original on 2012-05-22. Retrieved 2012-06-22. ^ Multiple (wiki). "Web application framework". Docforge. Archived from the original on 2020-06-20. Retrieved 2010-03-06. HTML5 Draft recommendation, changes to HTML and related APIs to ease authoring of web-based applications. Web Applications Working Group at the World Wide Web Consortium (W3C) PWAs on Web.dev by Google Developers. Retrieved from " 2Version 2 of the Hypertext Transfer Protocol used by the World Wide Web HTTP/2International standardRFC 9113Developed byIETFIntroducedMay 14, 2015; 10 years ago (2015-05-14)Superseded byHTTP/3Website HTTP/2 (originally named HTTP/2.0) is a major revision of the HTTP network protocol used by the World Wide Web. It was derived from the earlier experimental SPDY protocol, originally developed by Google.[1][2] HTTP/2 was developed by the HTTP Working Group (also called httpbis, where "bis" means "twice") of the Internet Engineering Task Force (IETF).[3][4][5] HTTP/2 is the first new version of HTTP since HTTP/1.1, which was standardized in RFC 2068 in 1997. The Working Group presented HTTP/2 to the Internet Engineering Steering Group (IESG) for consideration as a Proposed Standard in December 2014,[6][7] and IESG approved it to publish as Proposed Standard on February 17, 2015 (and was updated in February 2020 in regard to TLS 1.3 and again in June 2022). The initial HTTP/2 specification was published as on May 14, 2015.[8] The standardization effort was supported by Chrome, Opera, Firefox, Internet Explorer 11, Safari, Amazon Silk, and Edge browsers. Most major browsers had added HTTP/2 support by the end of 2015.[9] About 97% of web browsers used have the capability (and 100% of "tracked desktop" web browsers).[9] As of July 2023[update], 36% (after topping out at just over 50%) of the top 10 million websites support HTTP/2.[10] Its successor is HTTP/3, a major revision that builds on the concepts established by HTTP/2.[2][11][9][12] The working group charter mentions several goals and issues of concern:[4] Create a negotiation mechanism that allows clients and servers to elect to use HTTP/1.1, 2.0, or potentially other non-HTTP protocols. Maintain high-level compatibility with HTTP/1.1 (for example with methods, status codes, URIs, and most header fields). Decrease latency to improve page load speed in web browsers by considering: data compression of HTTP headers HTTP/2 Server Push prioritization of requests multiplexing multiple requests over a single TCP connection (fixing the HTTP-transaction-level head-of-line blocking problem in HTTP 1.x) Support common existing use cases of HTTP, such as desktop web browsers, mobile web browsers, web APIs, web servers at various scales, proxy servers, reverse proxy servers, firewalls, and content delivery networks. The proposed changes do not require any changes to how existing web applications work, but new applications can take advantage of new features for increased speed.[13] HTTP/2 leaves all of HTTP/1.1's high-level semantics, such as methods, status codes, header fields, and URIs, the same. What is new is how the data is framed and transported between the client and the server.[13] Websites that are efficient minimize the number of requests required to render an entire page by minifying (reducing the amount of code and packing smaller pieces of code into bundles, without reducing its ability to function) resources such as images and scripts. However, minification is not necessarily convenient nor efficient and may still require separate HTTP connections to get the page and the minified resources. HTTP/2 allows the server to "push" content, that is, to respond with data for more queries than the client requested. This allows the server to supply data it knows a web browser will need to render a web page, without waiting for the browser to examine the first response, and without the overhead of an additional request cycle.[14] Additional performance improvements in the first draft of HTTP/2 (which was a copy of SPDY) come from multiplexing of requests and responses to avoid some of the head-of-line blocking problem in HTTP 1 (even when HTTP pipelining is used), header compression, and prioritization of requests.[15] However, as HTTP/2 runs on top of a single TCP connection there is still potential for head-of-line blocking to occur if TCP packets are lost or delayed in transmission.[16] HTTP/2 no longer supports HTTP/1.1's chunked transfer encoding mechanism, as it provides its own, more efficient, mechanisms for data streaming.[17] SPDY (pronounced like "speedy") was a previous HTTP-replacement protocol developed by a research project spearheaded by Google.[18] Primarily focused on reducing latency, SPDY uses the same TCP pipe but different protocols to accomplish this reduction. The basic changes made to HTTP/1.1 to create SPDY included "true request pipelining without FIFO restrictions, message framing mechanism to simplify client and server development, mandatory compression (including headers), priority scheduling, and even bi-directional communication".[19] The HTTP Working Group considered Google's SPDY protocol, Microsoft's HTTP Speed+Mobility proposal (SPDY based),[18] and Network-Friendly HTTP Upgrade.[20] In July 2012, Facebook provided feedback on each of the proposals and recommended HTTP/2 be based on SPDY.[21] The initial draft of HTTP/2 was published in November 2012 and was based on a straight copy of SPDY.[22] The biggest difference between HTTP/1.1 and SPDY was that each user action in SPDY is given a "stream ID", meaning there is a single TCP channel connecting the user to the server. SPDY split requests into either control or data, using a "simple to parse binary protocol with two types of frames".[19][23] SPDY showed evident improvement over HTTP, with a new page load speedup ranging from 11% to 47%.[24] The development of HTTP/2 used SPDY as a jumping-off point. Among the many detailed differences between the protocols, the most notable is that HTTP/2 uses a fixed Huffman code-based header compression algorithm, instead of SPDY's dynamic stream-based compression. This helps to reduce the potential for compression oracle attacks on the protocol, such as the CRIME attack.[23] On February 9, 2015, Google announced plans to remove support for SPDY in Chrome in favor of support for HTTP/2.[25] This took effect starting with Chrome 51.[26][27] Date Milestone[4] December 20, 2007[28][29] First HTTP/1.1 Revision Internet Draft January 23, 2008[30] First HTTP Security Properties Internet Draft Early 2012[31] Call for Proposals for HTTP 2.0 October 14 – November 25, 2012[32][33] Working Group Last Call for HTTP/1.1 Revision November 28, 2012[34][35] First WG draft of HTTP 2.0, based upon draft-mbelshe-httpbis-spdy-00 Held/Eliminated Working Group Last Call for HTTP Security Properties September 2013[36][37] Submit HTTP/1.1 Revision to IESG for consideration as a Proposed Standard February 12, 2014[38] IESG approved HTTP/1.1 Revision to publish as a Proposed Standard June 6, 2014[28][39] Publish HTTP/1.1 Revision as RFC 7230, 7231, 7232, 7233, 7234, 7235 August 1, 2014 – September 1, 2014[7][40] Working Group Last call for HTTP/2 December 16, 2014[6] Submit HTTP/2 to IESG for consideration as a Proposed Standard December 31, 2014 – January 14, 2015[41] IETF Last Call for HTTP/2 January 22, 2015[42] IESG telechat to review HTTP/2 as Proposed Standard February 17, 2015[43] IESG approved HTTP/2 to publish as Proposed Standard May 14, 2015[44] Publish HTTP/2 as RFC 7540 February 2020 RFC 8740: HTTP/2 with TLS 1.3 June 2022 RFC 9113: Further refinements April 2024 DOS issues with CONTINUATION frames HTTP/2 is defined both for HTTP URIs (i.e. without TLS encryption, a configuration which is abbreviated in h2c) and for HTTPS URIs (over TLS using ALPN extension[45] where TLS 1.2 or newer is required, a configuration which is abbreviated in h2). Although the standard itself does not require usage of encryption,[46] all major client implementations (Firefox,[47] Chrome, Safari, Opera, IE, Edge) have stated that they will only support HTTP/2 over TLS, which makes encryption de facto mandatory.[48] The FreeBSD and Varnish developer Poul-Henning Kamp asserts that the standard was prepared on an unrealistically short schedule, ruling out any basis for the new HTTP/2 other than the SPDY protocol and resulting in other missed opportunities for improvement. Kamp criticizes the protocol itself for being inconsistent and having needless, overwhelming complexity. He also states that the protocol violates the protocol layering principle, for example by duplicating flow control that belongs in the transport layer (TCP). He also suggested that the new protocol should have removed HTTP Cookies, introducing a breaking change.[49] Initially, some members[who?] of the Working Group tried to introduce an encryption requirement in the protocol. This faced criticism. Critics stated that encryption has non-negligible computing costs and that many HTTP applications actually have no need for encryption and their providers have no desire to spend additional resources on it. Encryption proponents have stated that this encryption overhead is negligible in practice.[50] Poul-Henning Kamp has criticized the IETF for hastily standardizing Google's SPDY prototype as HTTP/2 due to political considerations.[49][51][52] The criticism of the agenda of mandatory encryption within the existing certificate framework is not new, nor is it unique to members of the open-source community – a Cisco employee stated in 2013 that the present certificate model is not compatible with small devices like routers, because the present model requires not only annual enrollment and remission of non-trivial fees for each certificate, but must be continually repeated on an annual basis.[53] In the end the Working Group did not reach consensus over the mandatory encryption,[46] although most client implementations require it, which makes encryption a de facto requirement. The HTTP/2 protocol also faced criticism for not supporting opportunistic encryption, a measure against passive monitoring similar to the STARTTLS mechanism that has long been available in other Internet protocols like SMTP. Critics have stated that the HTTP/2 proposal goes in violation of IETF's own RFC 7258 "Pervasive Monitoring Is an Attack", which also has a status of Best Current Practice 188.[54] RFC7258/BCP188 mandates that passive monitoring be considered as an attack, and protocols designed by IETF should take steps to protect against passive monitoring (for example, through the use of opportunistic encryption). A number of specifications for opportunistic encryption of HTTP/2 have been provided,[55][56][57] of which draft-nottingham-http2-encryption was adopted as an official work item of the working group, leading to the publication of RFC 8164 in May 2017. Although the design of HTTP/2 effectively addresses the HTTP-transaction-level head-of-line blocking problem by allowing multiple concurrent HTTP transactions, all those transactions are multiplexed over a single TCP connection, meaning that any packet-level head-of-line blocking of the TCP stream simultaneously blocks all transactions being accessed via that connection. This head-of-line blocking in HTTP/2 is now widely regarded as a design flaw, and much of the effort behind QUIC and HTTP/3 has been devoted to reduce head-of-line blocking issues.[58][59] Main article: Comparison of web server software The following web servers support HTTP/2: Apache httpd 2.4.12 supports HTTP/2 via the module mod_h2,[60] although appropriate patches must be applied to the source code of the server in order for it to support that module. As of Apache 2.4.17 all patches are included in the main Apache source tree, although the module itself was renamed mod_http2.[61] Old versions of SPDY were supported via the module mod_spdy,[62] however the development of the mod_spdy module has stopped.[63] Apache Tomcat 8.5 (requires a configuration change)[64] Apache Traffic Server[65] Caddy[66] Charles Proxy since version Charles 4.[67] Citrix NetScaler 11.x[68] Sucuri[69] F5 BIG-IP Local Traffic Manager 11.6[70] Barracuda Networks WAF (Web Application Firewall)[71] h2o (built from the ground up for HTTP/2 support)[72] HAProxy 1.8[73] Jetty 9.3[74] lighttpd 1.4.56[75] LiteSpeed Web Server 5.0[76] Microsoft IIS (in Windows 10,[77] Windows Server 2016, and Windows Server 2019) Netty 4.1[78] nghttpd (exclusively implements HTTP/2) nginx 1.9.5[79] released on September 22, 2015, using module ngx_http_v2_module and HTTP/2 Server Push since version 1.13.9 on February 20, 2018.[80] Node.js 8.13.0[81] (A separate module is available for Node.js 5.0[82] and Node 8.4 introduced experimental built-in support for HTTP/2.[83]) Kestrel web server for ASP.NET Core supports HTTP/2 since .NET Core 2.2.0-preview 1.[84] OpenLiteSpeed 1.3.11 and 1.4.8[85] Proxygen Pulse Secure Virtual Traffic Manager 10.2[86] Radware Alteon NG[87] ShimmerCat[88] Vert.x 3.3 Warp (Haskell web server, used by default in Yesod) Wildfly 9 Envoy proxy Akamai was the first major CDN to support HTTP/2 and HTTP/2 Server Push. Microsoft Azure supports HTTP/2. PageCDN supports HTTP/2 out of the box and provides user-interface to setup HTTP/2 Server Push in CDN dashboard.[89] CDN77 supports HTTP/2 using nginx (August 20, 2015). Cloudflare supports HTTP/2 using nginx with SPDY as a fallback for browsers without support, whilst maintaining all security and performance services.[90] Cloudflare was the first major CDN to support HTTP/2 Server Push.[91] AWS CloudFront supports HTTP/2[92] since September 7, 2016. Fastly supports HTTP/2 including Server Push.[93] Imperva Incapsula CDN supports HTTP/2.[94] The implementation includes support for WAF and DDoS mitigation features as well. KeyCDN supports HTTP/2 using nginx (October 6, 2015). HTTP/2 Test is a test page to verify if your server supports HTTP/2. BrandSSL supports HTTP/2. Voxility supports HTTP/2 using nginx since July, 2016. The implementation comes in support for Cloud DDoS mitigation services.[95] StackPath supports HTTP/2. Other implementations are collected on the GitHub HTTP/2 wiki. gRPC HTTP pipelining HTTP request and response messages HTTP/3 QUIC SPDY WebSocket Web Server Web Browser Comparison of web browsers § Protocol support ^ Bright, Peter (February 18, 2015). "HTTP/2 finished, coming to browsers within weeks". Ars Technica. Archived from the original on March 30, 2019. ^ a b Cimpanu, Catalin (November 12, 2018). "HTTP-over-QUIC to be renamed HTTP/3". ZDNet. Retrieved November 19, 2018. ^ Thomson, M.; Belshe, M.; Peon, R. (November 29, 2014). "Hypertext Transfer Protocol version 2: draft-ietf-httpbis-http2-16". Ietf Datatracker. HTTPbis Working Group. Retrieved February 11, 2015. ^ a b c "HTTP (httpbis)". Internet Engineering Task Force Datatracker. Archived from the original on January 6, 2024. ^ "IETF HTTP Working Group". httpwg.org. Retrieved December 15, 2019. ^ a b "History for draft-ietf-httpbis-http2-16". IETF. Retrieved January 3, 2015. 2014-12-16 IESG state changed to Publication Requested ^ a b Raymor, Brian (August 6, 2014). "Wait for it – HTTP/2 begins Working Group Last Call!". Microsoft Open Technologies. Archived from the original on October 6, 2014. Retrieved October 17, 2018. ^ Belshe, M.; Peon, R.; Thomson, M. (May 2015). Thomson, M (ed.). "RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2)". IETF. doi:10.17487/RFC7540. Retrieved May 14, 2015. ^ a b c ""HTTP/2" | Can I use... Support tables for HTML5, CSS3, etc". canIuse.com. Retrieved April 3, 2023. ^ "Usage of HTTP/2 for websites". World Wide Web Technology Surveys. W3Techs. Retrieved July 10, 2023. ^ Bishop, Mike (July 9, 2019). "Hypertext Transfer Protocol Version 3 (HTTP/3)". Ietf Datatracker. Retrieved July 31, 2019. ^ Cimpanu, Catalin (26 September 2019). "Cloudflare, Google Chrome, and Firefox add HTTP/3 support". ZDNet. Retrieved 27 September 2019. ^ a b Ilya Grigorik. "Chapter 12: HTTP 2.0". High Performance Browser Networking. O'Reilly Media, Inc. HTTP/2 does not modify the application semantics of HTTP in any way ^ Pratt, Michael. "Apiux". apiux.com. Retrieved March 19, 2014. ^ Dio Synodinos (November 2012). "HTTP 2.0 First Draft Published". InfoQ.com. C4Media Inc. ^ Javier Garza (October 2017). "How does HTTP/2 solve the Head of Line blocking (HOL) issue". ^ Belshe, Mike; Thomson, Martin; Peon, Roberto (May 2015). Thomson, M. (ed.). "Hypertext Transfer Protocol Version 2 (HTTP/2)". tools.ietf.org. doi:10.17487/RFC7540. Retrieved November 17, 2017. HTTP/2 uses DATA frames to carry message payloads. The "chunked" transfer encoding defined in Section 4.1 of [RFC7230] MUST NOT be used in HTTP/2 ^ a b Sebastian Anthony (March 28, 2012). "S&M vs. SPDY: Microsoft and Google battle over the future of HTTP 2.0". ExtremeTech. ^ a b Grigorik, Ilya. "Life beyond HTTP 1.1: Google's SPDY". ^ Willy Tarreau; Amos Jeffries; Adrien de Croy; Poul-Henning Kamp (March 29, 2012). "Proposal for a Network-Friendly HTTP Upgrade". Network Working Group. Internet Engineering Task Force. ^ Doug Beaver (July 15, 2012). "HTTP2 Expression of Interest" (mailing list). W3C. ^ Dio Synodinos (November 30, 2012). "HTTP/2 First Draft Published". InfoQ. ^ a b Ilya, Grigorik (2015). HTTP/2 : a new excerpt from high performance browser networking (May 2015, First ed.). Sebastopol, Calif.: O'Reilly Media. pp. 211–224. ISBN 9781491932483. OCLC 1039459460. ^ "SPDY: An experimental protocol for a faster web". The Chromium Projects. ^ Chris Bentzel; Bence Béky (February 9, 2015). "Hello HTTP/2, Goodbye SPDY". Chromium Blog. Update: To better align with Chrome's release cycle, SPDY and NPN support will be removed with the release of Chrome 51. ^ "API Deprecations and Removals in Chrome 51". TL;DR: Support for HTTP/2 is widespread enough that SPDY/3.1 support can be dropped. ^ Shadrin, Nick (June 7, 2016). "Supporting HTTP/2 for Google Chrome Users | NGINX". NGINX. Retrieved July 10, 2017. ^ a b Nottingham, Mark (June 7, 2014). "RFC2616 is Dead". Retrieved September 20, 2014. ^ "HTTP/1.1, part 1: URIs, Connections, and Message Parsing: draft-ietf-httpbis-p1-messaging-00". December 20, 2007. Retrieved September 20, 2014. ^ "Security Requirements for HTTP: draft-ietf-httpbis-security-properties-00.txt". January 23, 2008. Retrieved September 20, 2014. ^ Nottingham, Mark (January 24, 2012). "Rechartering HTTPbis". Retrieved September 20, 2014. ^ Nottingham, Mark (October 14, 2012). "Working Group Last Call for HTTP/1.1 p1 and p2". Retrieved September 20, 2014. ^ Nottingham, Mark (October 23, 2012). "Second Working Group Last Call for HTTP/1.1 p4 to p7". Retrieved September 20, 2014. ^ "SPDY Protocol: draft-ietf-httpbis-http2-00". HTTPbis Working Group. November 28, 2012. Retrieved September 20, 2014. ^ Nottingham, Mark (November 30, 2012). "First draft of HTTP/2". Retrieved September 20, 2014. ^ Fielding, Roy T.; Reschke, Julian (June 6, 2014). "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing". Archived from the original on August 13, 2014. Retrieved September 20, 2014. ^ "Last Call: (Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing) to Proposed Standard". The IESG. October 21, 2013. Retrieved September 20, 2014. ^ "Protocol Action: 'Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing' to Proposed Standard (draft-ietf-httpbis-p1-messaging-26.txt)". ietf-announce (Mailing list). The IESG. February 12, 2014. Retrieved January 18, 2015. ^ The RFC Editor Team (June 6, 2014). "RFC 7230 on Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing". ietf-announce (Mailing list). Retrieved January 18, 2015. ^ Nottingham, Mark (August 1, 2014). "Working Group Last Call: draft-ietf-httpbis-http2-14 and draft-ietf-httpbis-header-compression-09". HTTP Working Group. Retrieved September 7, 2014. ^ "Last Call: (Hypertext Transfer Protocol version 2) to Proposed Standard from The IESG on 2014-12-31". Internet Engineering Task Force. 2014. Retrieved January 1, 2015. ^ "IESG Agenda: 2015-01-22". IETF. Archived from the original on January 15, 2015. Retrieved January 15, 2015. ^ The IESG (February 17, 2015). "Protocol Action: 'Hypertext Transfer Protocol version 2' to Proposed Standard (draft-ietf-httpbis-http2-17.txt)". httpbis (Mailing list). Retrieved February 18, 2015. ^ The RFC Editor Team (May 14, 2015). "RFC 7540 on Hypertext Transfer Protocol Version 2 (HTTP/2)". ietf-announce (Mailing list). ^ Friedl, S; Popov, A.; Langley, A.; Stephan, E. (July 2014). "RFC 7301 - Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension". IETF. doi:10.17487/RFC7301. ^ a b "HTTP/2 Frequently Asked Questions". IETF HTTP Working Group. Retrieved September 8, 2014. ^ "Networking/http2". MozillaWiki. Retrieved September 7, 2014. ^ "HTTP/2 Implementation Status". mnot’s blog. ^ a b Kamp, Poul-Henning (January 6, 2015). "HTTP/2.0 – The IETF is Phoning It In (Bad protocol, bad politics)". ACM Queue. Vol. 13, no. 2. pp. 10–12. doi:10.1145/2732266.2716278. ISSN 1542-7730. ^ Grigorik, Ilya. "Is TLS Fast Yet?". Retrieved December 30, 2015. ^ Kamp, Poul-Henning (2015). "Http/2.0". Communications of the ACM. 58 (3): 40. doi:10.1145/2717515. S2CID 20337779. ^ Kamp, Poul-Henning (January 7, 2015). "Re: Last Call: (Hypertext Transfer Protocol version 2) to Proposed Standard". ietf-http-wg@w3.org (Mailing list). Retrieved January 12, 2015. ^ Lear, Eliot (August 25, 2013). "Mandatory encryption *is* theater". ietf-http-wg@w3.org (Mailing list). Retrieved January 26, 2015. ^ Murenin, Constantine A. (January 9, 2015). "Re: Last Call: (Hypertext Transfer Protocol version 2) to Proposed Standard". ietf-http-wg@w3.org (Mailing list). Retrieved January 12, 2015. ^ Paul Hoffman. "Minimal Unauthenticated Encryption (MUE) for HTTP-2: draft-hoffman-httpbis-minimal-unauth-enc-01". Internet Engineering Task Force. ^ Mark Nottingham; Martin Thomson. "Opportunistic Encryption for HTTP URIs: draft-nottingham-http2-encryption-03". Internet Engineering Task Force. ^ Mark Nottingham; Martin Thomson. "Opportunistic Security for HTTP: draft-ietf-httpbis-http2-encryption-01". Ietf Datatracker. Internet Engineering Task Force. ^ Huston, Geoff (March 4, 2019). "A Quick Look at QUIC". www.circleid.com. Retrieved August 2, 2019. ^ Gal, Shauli (June 22, 2017). "The Full Picture on HTTP/2 and HOL Blocking". Medium. Retrieved August 3, 2019. ^ "http2 module for apache httpd". Retrieved July 28, 2015. ^ "Apache 2.4.17 release changelog". Retrieved August 22, 2017. ^ Matthew Steele (June 19, 2014). "mod_spdy is now an Apache project". Google Developers Blog. ^ "Log of /httpd/mod_spdy". svn.apache.org. Retrieved February 3, 2017. ^ "Apache Tomcat Migration". Retrieved July 29, 2016. ^ "Apache Traffic Server Downloads". trafficserver.apache.org. September 21, 2015. ^ Server, Caddy Web (March 23, 2016). "Caddy 2 - The Ultimate Server with Automatic HTTPS". caddyserver.com. Retrieved August 8, 2020. ^ "Charles 4 has HTTP/2". Public Object. August 2, 2016. Retrieved October 12, 2020. ^ "3 Simple Steps to Bring HTTP/2 Performance to Legacy Web Applications". September 22, 2015. ^ "Sucuri += HTTP/2 — Announcing HTTP/2 Support". Sucuri. November 27, 2015. Retrieved December 5, 2015. ^ Robert Haynes. "Goodbye SPDY, Hello HTTP/2". F5 Networks. Retrieved September 18, 2015. ^ Risov Chakrabortty (July 5, 2016). "New features, capabilities added to Barracuda Web Application Firewall". Barracuda Networks. ^ "H2O - the optimized HTTP/2 server". h2o.examp1e.net. ^ "What's New in HAProxy 1.8". haproxy.com. November 2017. Retrieved February 9, 2018. ^ "Jetty change log". Eclipse Foundation. May 28, 2015. Retrieved May 28, 2015. ^ "Feature #2813: Support for HTTP/2 protocol", Lighttpd ^ "LSWS 5.0 Is Out – Support for HTTP/2, ESI, LiteMage Cache". April 17, 2015. ^ Rob Trace; David Walp (October 8, 2014). "HTTP/2: The Long-Awaited Sequel". MSDN IEBlog. Microsoft Corporation. ^ "Netty.news: Netty 4.1.0.Final released". netty.io. Retrieved June 1, 2016. ^ "nginx changelog". www.nginx.com. September 22, 2015. ^ "Changes with nginx 1.14.2". nginx.org. December 4, 2018. Retrieved September 27, 2019. ^ Foundation, Node js (November 20, 2018). "Node v8.13.0 (LTS)". Node.js. Retrieved June 5, 2019. ^ "Node http2". www.github.com. July 26, 2016. ^ "Node v8.4.0 (Current)". nodejs.org. August 15, 2017. ^ "ASP.NET Core 2.2.0-preview1: HTTP/2 in Kestrel". Retrieved April 6, 2021. ^ "OpenLiteSpeed 1.4.5 change log". LiteSpeed Technologies, Inc. February 26, 2015. Retrieved February 26, 2015. ^ "Pulse Virtual Traffic Manager". August 22, 2017. ^ "Radware Combines an Integrated HTTP/2 Gateway with its Leading Fastview Technology to Provide Web Server Platforms Increased Acceleration". July 20, 2015. ^ "www.shimmercat.com". March 23, 2016. Archived from the original on March 31, 2022. Retrieved March 23, 2016. ^ "Why PageCDN, and what problem does it solve?". PageCDN. Retrieved January 11, 2020. ^ "HTTP/2 is here! Goodbye SPDY? Not quite yet". CloudFlare. Retrieved December 5, 2015. ^ Krasnov, Vlad (April 28, 2016). "Announcing Support for HTTP/2 Server Push". CloudFlare. Retrieved May 18, 2016. ^ "Amazon CloudFront now supports HTTP/2". Amazon Web Services, Inc. Retrieved September 8, 2016. ^ "Announcing Limited Availability for HTTP/2". June 30, 2016. Retrieved August 22, 2017. ^ "HTTP/2 is here: What You Need to Know". Retrieved November 1, 2015. ^ "HTTP/2 more at risk to cyber attacks?". Information Age. August 3, 2016. Retrieved February 4, 2019. Official website HTTP/2 on GitHub RFC 7540 – Hypertext Transfer Protocol version 2 (HTTP/2) RFC 7541 – HPACK: Header Compression for HTTP/2 HTTP/2 explained (Daniel Stenberg) SPDY Protocol (draft-mbelshe-httpbis-spdy-00) HTTP Speed+Mobility (draft-Montenegro-httpbis-speed-mobility-01) Proposal for a Network-Friendly HTTP Upgrade (draft-tarreau-httpbis-network-friendly-00) Retrieved from " 3Layer 7 network protocol published in 2022 HTTP/3International standardRFC 9114[1] (HTTP/3 also uses the completed QUIC protocol described in RFC 9000 and related RFCs such as RFC 9001)Developed byIETFIntroducedJune 2022Website Internet history timeline Early research and development: 1960–1964: RAND networking concepts developed 1962–1964: ARPA networking ideas 1965 (1965): NPL network concepts conceived 1966 (1966): Merit Network founded 1967 (1967): ARPANET planning begins 1967 (1967): Symposium on Operating Systems Principles 1969 (1969): NPL followed by the ARPANET carry their first packets 1970 (1970): Network Information Center (NIC) 1971 (1971): Tymnet switched-circuit network 1972 (1972): Merit Network's packet-switched network operational 1972 (1972): Internet Assigned Numbers Authority (IANA) established 1973 (1973): CYCLADES network demonstrated 1973 (1973): PARC Universal Packet development begins 1974 (1974): Transmission Control Program specification published 1975 (1975): Telenet commercial packet-switched network 1976 (1976): X.25 protocol approved and deployed on public data networks 1978 (1978): Minitel introduced 1979 (1979): Internet Activities Board (IAB) 1980 (1980): USENET news using UUCP 1980 (1980): Ethernet standard introduced 1981 (1981): BITNET established Merging the networks and creating the Internet: 1981 (1981): Computer Science Network (CSNET) 1982 (1982): TCP/IP protocol suite formalized 1982 (1982): Simple Mail Transfer Protocol (SMTP) 1983 (1983): Domain Name System (DNS) 1983 (1983): MILNET split off from ARPANET 1984 (1984): OSI Reference Model released 1985 (1985): First .COM domain name registered 1986 (1986): NSFNET with 56 kbit/s links 1986 (1986): Internet Engineering Task Force (IETF) 1987 (1987): UUNET founded 1988 (1988): NSFNET upgraded to 1.5 Mbit/s (T1) 1988 (1988): Morris worm 1988 (1988): Complete Internet protocol suite 1989 (1989): Border Gateway Protocol (BGP) 1989 (1989): PSINet founded, allows commercial traffic 1989 (1989): Federal Internet Exchanges (FIX East|FIXes) 1990 (1990): GOSIP (without TCP/IP) 1990 (1990): ARPANET decommissioned 1990 (1990): Advanced Network and Services (ANS) 1990 (1990): UUNET/Alternet allows commercial traffic 1990 (1990): Archie search engine 1991 (1991): Wide area information server (WAIS) 1991 (1991): Gopher 1991 (1991): Commercial Internet eXchange (CIX) 1991 (1991): ANS CO+RE allows commercial traffic 1991 (1991): World Wide Web (WWW) 1992 (1992): NSFNET upgraded to 45 Mbit/s (T3) 1992 (1992): Internet Society (ISOC) established 1993 (1993): Classless Inter-Domain Routing (CIDR) 1993 (1993): InterNIC established 1993 (1993): AOL added USENET access 1993 (1993): Mosaic web browser released 1994 (1994): Full text web search engines 1994 (1994): North American Network Operators' Group (NANOG) established Commercialization, privatization, broader access leads to the modern Internet: 1995 (1995): New Internet architecture with commercial ISPs connected at NAPs 1995 (1995): NSFNET decommissioned 1995 (1995): GOSIP updated to allow TCP/IP 1995 (1995): very high-speed Backbone Network Service (vBNS) 1995 (1995): IPv6 proposed 1996 (1996): AOL changes pricing model from hourly to monthly 1998 (1998): Internet Corporation for Assigned Names and Numbers (ICANN) 1999 (1999): IEEE 802.11b wireless networking 1999 (1999): Internet2/Abilene Network 1999 (1999): vBNS+ allows broader access 2000 (2000): Dot-com bubble bursts 2001 (2001): New top-level domain names activated 2001 (2001): Code Red I, Code Red II, and Nimda worms 2003 (2003): UN World Summit on the Information Society (WSIS) phase I 2003 (2003): National LambdaRail founded 2004 (2004): UN Working Group on Internet Governance (WGIG) 2005 (2005): UN WSIS phase II 2006 (2006): First meeting of the Internet Governance Forum 2010 (2010): First internationalized country code top-level domains registered 2012 (2012): ICANN begins accepting applications for new generic top-level domain names 2013 (2013): Montevideo Statement on the Future of Internet Cooperation 2014 (2014): NetMundial international Internet governance proposal 2016 (2016): ICANN contract with U.S. Dept. of Commerce ends, IANA oversight passes to the global Internet community on October 1st Examples of Internet services: 1989 (1989): AOL dial-up service provider, email, instant messaging, and web browser 1990 (1990): IMDb Internet movie database 1994 (1994): Yahoo! web directory 1995 (1995): Amazon online retailer 1995 (1995): eBay online auction and shopping 1995 (1995): Craigslist classified advertisements 1995 (1995): AltaVista search engine 1996 (1996): Outlook (formerly Hotmail) free web-based e-mail 1996 (1996): RankDex search engine 1997 (1997): Google Search 1997 (1997): Babel Fish automatic translation 1998 (1998): Yahoo Groups (formerly Yahoo! Clubs) 1998 (1998): PayPal Internet payment system 1998 (1998): Rotten Tomatoes review aggregator 1999 (1999): 2ch Anonymous textboard 1999 (1999): i-mode mobile internet service 1999 (1999): Napster peer-to-peer file sharing 2000 (2000): Baidu search engine 2001 (2001): 2channel Anonymous textboard 2001 (2001): BitTorrent peer-to-peer file sharing 2001 (2001): Wikipedia, the free encyclopedia 2003 (2003): LinkedIn business networking 2003 (2003): Myspace social networking site 2003 (2003): Skype Internet voice calls 2003 (2003): iTunes Store 2003 (2003): 4chan Anonymous imageboard 2003 (2003): The Pirate Bay, torrent file host 2004 (2004): Facebook social networking site 2004 (2004): Podcast media file series 2004 (2004): Flickr image hosting 2005 (2005): YouTube video sharing 2005 (2005): Reddit link voting 2005 (2005): Google Earth virtual globe 2006 (2006): Twitter microblogging 2007 (2007): WikiLeaks anonymous news and information leaks 2007 (2007): Google Street View 2007 (2007): Kindle, e-reader and virtual bookshop 2008 (2008): Amazon Elastic Compute Cloud (EC2) 2008 (2008): Dropbox cloud-based file hosting 2008 (2008): Encyclopedia of Life, a collaborative encyclopedia intended to document all living species 2008 (2008): Spotify, a DRM-based music streaming service 2009 (2009): Bing search engine 2009 (2009): Google Docs, Web-based word processor, spreadsheet, presentation, form, and data storage service 2009 (2009): Kickstarter, a threshold pledge system 2009 (2009): Bitcoin, a digital currency 2010 (2010): Instagram, photo sharing and social networking 2011 (2011): Google+, social networking 2011 (2011): Snapchat, photo sharing 2012 (2012): Coursera, massive open online courses 2016 (2016): TikTok, video sharing and social networking HTTP/3 is the third major version of the Hypertext Transfer Protocol used to exchange information on the World Wide Web, complementing the widely deployed HTTP/1.1 and HTTP/2. Unlike previous versions which relied on the well-established TCP (published in 1974),[2] HTTP/3 uses QUIC (officially introduced in 2021),[3] a multiplexed transport protocol built on UDP.[4] HTTP/3 uses similar semantics compared to earlier revisions of the protocol, including the same request methods, status codes, and message fields, but encodes them and maintains session state differently. However, partially due to the protocol's adoption of QUIC, HTTP/3 has lower latency and loads more quickly in real-world usage when compared with previous versions: in some cases over four times as fast than with HTTP/1.1 (which, for many websites, is the only HTTP version deployed).[5][6] As of September 2024, HTTP/3 is supported by more than 95% of major web browsers in use[7] and 34% of the top 10 million websites.[8] It has been supported by Chromium (and derived projects including Google Chrome, Microsoft Edge, Samsung Internet, and Opera)[9] since April 2020 and by Mozilla Firefox since May 2021.[7][10] Safari 14 implemented the protocol but it remains disabled by default.[11] Protocol stack of HTTP/3 compared to HTTP/1.1 and HTTP/2 HTTP/3 originates from an Internet Draft adopted by the QUIC working group. The original proposal was named "HTTP/2 Semantics Using The QUIC Transport Protocol",[12] and later renamed "Hypertext Transfer Protocol (HTTP) over QUIC".[13] On 28 October 2018 in a mailing list discussion, Mark Nottingham, Chair of the IETF HTTP and QUIC Working Groups, proposed renaming HTTP-over-QUIC to HTTP/3, to "clearly identify it as another binding of HTTP semantics to the wire protocol [...] so people understand its separation from QUIC".[14] Nottingham's proposal was accepted by fellow IETF members a few days later. The HTTP working group was chartered to assist the QUIC working group during the design of HTTP/3, then assume responsibility for maintenance after publication.[15] Support for HTTP/3 was added to Chrome (Canary build) in September 2019 and then eventually reached stable builds, but was disabled by a feature flag. It was enabled by default in April 2020.[9] Firefox added support for HTTP/3 in November 2019 through a feature flag[7][16][17] and started enabling it by default in April 2021 in Firefox 88.[7][10] Experimental support for HTTP/3 was added to Safari Technology Preview on April 8, 2020[18] and was included with Safari 14 that ships with iOS 14 and macOS 11,[11][19] but it's still disabled by default as of Safari 16, on both macOS and iOS.[citation needed] On 6 June 2022, IETF published HTTP/3 as a Proposed Standard in RFC 9114.[1] HTTP semantics are consistent across versions: the same request methods, status codes, and message fields are typically applicable to all versions. The differences are in the mapping of these semantics to underlying transports. Both HTTP/1.1 and HTTP/2 use TCP as their transport. HTTP/3 uses QUIC, a transport layer network protocol which uses user space congestion control over the User Datagram Protocol (UDP). The switch to QUIC aims to fix a major problem of HTTP/2 called "head-of-line blocking": because the parallel nature of HTTP/2's multiplexing is not visible to TCP's loss recovery mechanisms, a lost or reordered packet causes all active transactions to experience a stall regardless of whether that transaction was impacted by the lost packet. Because QUIC provides native multiplexing, lost packets only impact the streams where data has been lost. The HTTPS DNS resource record as defined in RFC 9460[20] allows for connecting without first receiving the Alt-Svc header via previous HTTP versions, therefore removing the 1 RTT of handshaking of TCP.[21][22] There is client support for HTTPS resource records since Firefox 92, iOS 14, reported Safari 14 support, and Chromium supports it behind a flag.[23][24][25] Browser support for HTTP/3 Browser Version implemented (disabled by default) Version shipped (enabled by default) Comment Chrome Stable build (79) December 2019 87[7] April 2020[26] Earlier versions implemented other drafts of QUIC Edge Stable build (79) December 2019 87 April 2020 Edge 79 was the first version based on Chromium Firefox Stable build (72.0.1) January 2020 88[10] April 2021[27] Safari Stable build (14.0) September 2020 16.4 March 2023 Apple is testing HTTP/3 support on some Safari users starting with Safari 16.4.[28] Open-source libraries that implement client or server logic for QUIC and HTTP/3 include[29] Libraries implementing HTTP/3 Name Client Server Programming language Company Repository lsquic Yes Yes C LiteSpeed nghttp3 Yes Yes C h2o No Yes C libcurl[30][31] Yes No C MsQuic[32] Yes Yes C Microsoft proxygen Yes Yes C++ Facebook Cronet Yes Yes C++ Google .NET[33] Yes Yes C# (using MsQuic)[34] Microsoft quic-go Yes Yes Go http3 Yes Yes Haskell Kwik Yes Yes Java Flupke Yes Yes Java aioquic Yes Yes Python quiche Yes Yes Rust Cloudflare neqo Yes Yes Rust Mozilla quinn Yes Yes Rust s2n-quic Yes Yes Rust Amazon Web Services On 7 June 2021, LiteSpeed Web Server (and OpenLiteSpeed) 6.0.2 was released and became the first version to enable HTTP/3 by default.[35] Caddy web server v2.6.0 (released 20 September 2022) has HTTP/3 enabled by default.[36] Nginx supports HTTP/3 since 1.25.0 (released 23 May 2023). A technology preview of nginx with HTTP/3 support was released in June 2020.[37] Binary packages of nginx with HTTP/3 support have been released in February 2023.[38] Cloudflare distributes a patch for nginx that integrates the quiche HTTP/3 library into it.[39] Microsoft IIS support for HTTP/3 is enabled natively with Windows Server 2022/Windows 11.[40] HAProxy supports HTTP/3 over QUIC since version 2.6 released on 31 May 2022.[41][42] Nimble Streamer supports HTTP/3 since 4.1.8-1[43] for HTTP-based protocols. Internet portal Fast and Secure Protocol – Terminal command scheme used to transfer data ^ a b M. Bishop, ed. (June 2022). HTTP/3. Internet Engineering Task Force. doi:10.17487/RFC9114. ISSN 2070-1721. RFC 9114. Proposed Standard. ^ V. Cerf; Y. Dalal; C. Sunshine (December 1974). SPECIFICATION OF INTERNET TRANSMISSION CONTROL PROGRAM. Network Working Group. doi:10.17487/RFC0675. RFC 675. Obsolete. Obsoleted by RFC 7805. NIC 2. INWG 72. ^ J. Iyengar; M. Thomson, eds. (May 2021). QUIC: A UDP-Based Multiplexed and Secure Transport. Internet Engineering Task Force. doi:10.17487/RFC9000. ISSN 2070-1721. RFC 9000. Proposed Standard. ^ "What is HTTP/3?". Cloudflare. Archived from the original on 4 July 2022. Retrieved 12 July 2022. ^ Perna, Gianluca; Trevisan, Martino; Giordano, Danilo; Drago, Idilio (1 April 2022). "A first look at HTTP/3 adoption and performance". Computer Communications. 187: 115–124. doi:10.1016/j.comcom.2022.02.005. hdl:11368/3025202. ISSN 0140-3664. S2CID 246936473. ^ "HTTP/3 is Fast". Request Metrics. Retrieved 1 July 2022. ^ a b c d e ""HTTP/3" | Can I use... Support tables for HTML5, CSS3, etc". canIuse.com. Retrieved 11 August 2024. ^ "Usage of HTTP/3 for websites". World Wide Web Technology Surveys. W3Techs. Retrieved 11 August 2024. ^ a b "Enabling QUIC in tip-of-tree". groups.google.com. Retrieved 8 April 2021. ^ a b c Damjanovic, Dragana (16 April 2021). "QUIC and HTTP/3 Support now in Firefox Nightly and Beta". Mozilla Hacks – the Web developer blog. Retrieved 17 April 2021. ^ a b "Safari 14 Release Notes". developer.apple.com. Retrieved 4 December 2020. ^ Shade, Robbie (8 July 2016). HTTP/2 Semantics Using The QUIC Transport Protocol. IETF. I-D draft-shade-quic-http2-mapping. ^ Cimpanu, Catalin (12 November 2018). "HTTP-over-QUIC to be renamed HTTP/3". ZDNet. Retrieved 12 November 2018. ^ Nottingham, Mark (28 October 2018). "Identifying our deliverables". IETF Mail Archive. ^ "Hypertext Transfer Protocol Charter". ietf.org. Retrieved 2 September 2020. ^ Daniel, Stenberg. "Daniel Stenberg announces HTTP/3 support in Firefox Nightly". Twitter. Retrieved 5 November 2019. ^ Cimpanu, Catalin (26 September 2019). "Cloudflare, Google Chrome, and Firefox add HTTP/3 support". ZDNet. Retrieved 27 September 2019. ^ "Release Notes for Safari Technology Preview 104". webkit.org. 8 April 2020. Retrieved 7 August 2020. ^ Ng, Gary (23 June 2020). "Apple's Safari Adds Support for HTTP3 in iOS 14 and macOS 11". iphoneincanada.ca. Retrieved 25 June 2021. ^ Benjamin M. Schwartz; Mike Bishop; Erik Nygren (November 2023). Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records). Internet Engineering Task Force. doi:10.17487/RFC9460. ISSN 2070-1721. RFC 9460. Proposed Standard. ^ "HTTPS RR". MDN. Mozilla. Retrieved 25 October 2022. ^ Schwartz, Benjamin M.; Bishop, Mike; Nygren, Erik (12 June 2020). Service binding and parameter specification via the DNS. IETF. I-D draft-ietf-dnsop-svcb-https. ^ "Firefox 92 for developers". Mozilla Corporation. 7 September 2021. Retrieved 25 October 2022. ^ "Feature: HTTP->HTTPS redirect for HTTPS DNS records". Google Inc. Retrieved 25 October 2022. ^ Patrick Mevzek (24 August 2021). "What's the use case of SVCB (type 65, service binding) RR". Stack Exchange Inc. Retrieved 25 October 2022. ^ "Enabling QUIC in tip-of-tree". groups.google.com. Retrieved 9 April 2021. ^ "Firefox Release Owners - MozillaWiki". wiki.mozilla.org. Retrieved 9 April 2021. ^ Jen Simmons (4 April 2023). "HTTP/3 support shipped in Safari 14.0". GitHub. Retrieved 7 April 2023. ^ "QUIC Implementations". GitHub. Retrieved 8 April 2021. ^ "First HTTP/3 with curl". Daniel Stenberg. 5 August 2019. Retrieved 2 October 2019. ^ "HTTP3 (and QUIC)". Daniel Stenberg. 23 August 2023. Retrieved 27 August 2023. ^ "MsQuic is Open Source". 28 April 2020. Retrieved 28 April 2020. ^ "HTTP/3 support in .NET 6". 17 September 2021. Retrieved 17 September 2021. ^ "HTTP/3 support in .NET 6". .NET Blog. 17 September 2021. Retrieved 12 January 2022. ^ "LiteSpeed Web Server Release Log - LiteSpeed Technologies". www.litespeedtech.com. Retrieved 12 February 2022. Enable HTTP/3 v1 by default. ^ "Release 2.6.0 · caddyserver/caddy". Github. 22 September 2022. Retrieved 20 September 2022. ^ "Introducing a Technology Preview of NGINX Support for QUIC and HTTP/3". NGINX. 10 June 2020. Retrieved 11 June 2020. ^ "Binary Packages Now Available for the Preview NGINX QUIC+HTTP/3 Implementation". NGINX. 8 February 2023. Retrieved 30 March 2023. ^ "Experiment with HTTP/3 using NGINX and quiche". The Cloudflare Blog. 17 October 2019. Retrieved 9 November 2019. ^ Tratcher. "Use ASP.NET Core with HTTP/3 on IIS". docs.microsoft.com. Retrieved 29 April 2022. ^ "Announcing HAProxy 2.6". HAProxy Blog. 31 May 2022. ^ "QUIC Implementation in HAProxy". HAProxyConf video presentation. 25 January 2023. ^ "HTTP/3 and QUIC support in Nimble Streamer". NGINX. 14 February 2025. Retrieved 17 February 2025. Official website IETF QUIC Working Group on GitHub HTTP/3 explained (Daniel Stenberg) HTTP/3 on canIuse.com List of QUIC implementations on the IETF QUIC Working Group Wiki Retrieved from " 4Application layer protocol HTTPInternational standard RFC 1945 HTTP/1.0 RFC 9110 HTTP Semantics RFC 9111 HTTP Caching RFC 9112 HTTP/1.1 RFC 9113 HTTP/2 RFC 7541 HTTP/2: HPACK Header Compression RFC 8164 HTTP/2: Opportunistic Security for HTTP/2 RFC 8336 HTTP/2: The ORIGIN HTTP/2 Frame RFC 8441 HTTP/2: Bootstrapping WebSockets with HTTP/2 RFC 9114 HTTP/3 RFC 9204 HTTP/3: QPACK: Field Compression Developed byInitially CERN; IETF, W3CIntroduced1991; 34 years ago (1991)Websitehttpwg.org/specs/ HTTP Persistence Compression HTTPS QUIC Request methods OPTIONS GET HEAD POST PUT DELETE TRACE CONNECT PATCH Header fields Cookie ETag Location HTTP referer DNT X-Forwarded-For Response status codes 301 Moved Permanently 302 Found 303 See Other 403 Forbidden 404 Not Found 451 Unavailable for Legal Reasons Security access control methods Basic access authentication Digest access authentication Security vulnerabilities HTTP header injection HTTP request smuggling HTTP response splitting HTTP parameter pollution vte Internet protocol suite Application layer BGP DHCP (v6) DNS FTP HTTP (HTTP/3) HTTPS IMAP IPP IRC LDAP MGCP MQTT NNTP NTP OSPF POP PTP ONC/RPC RTP RTSP RIP SIP SMTP SNMP SSH Telnet TLS/SSL XMPP more... Transport layer TCP UDP DCCP SCTP RSVP QUIC more... Internet layer IP v4 v6 ICMP (v6) NDP ECN IGMP IPsec more... Link layer ARP Tunnels PPP MAC more... vte HTTP (Hypertext Transfer Protocol) is an application layer protocol in the Internet protocol suite model for distributed, collaborative, hypermedia information systems.[1] HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser. Development of HTTP was initiated by Tim Berners-Lee at CERN in 1989 and summarized in a simple document describing the behavior of a client and a server using the first HTTP version, named 0.9.[2] That version was subsequently developed, eventually becoming the public 1.0.[3] Development of early HTTP Requests for Comments (RFCs) started a few years later in a coordinated effort by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C), with work later moving to the IETF. HTTP/1 was finalized and fully documented (as version 1.0) in 1996.[4] It evolved (as version 1.1) in 1997 and then its specifications were updated in 1999, 2014, and 2022.[5] Its secure variant named HTTPS is used by more than 85% of websites.[6] HTTP/2, published in 2015, provides a more efficient expression of HTTP's semantics "on the wire". As of August 2024,[update] it is supported by 66.2% of websites[7][8] (35.3% HTTP/2 + 30.9% HTTP/3 with backwards compatibility) and supported by almost all web browsers (over 98% of users).[9] It is also supported by major web servers over Transport Layer Security (TLS) using an Application-Layer Protocol Negotiation (ALPN) extension[10] where TLS 1.2 or newer is required.[11][12] HTTP/3, the successor to HTTP/2, was published in 2022.[13] As of February 2024,[update] it is now used on 30.9% of websites[14] and is supported by most web browsers, i.e. (at least partially) supported by 97% of users.[15] HTTP/3 uses QUIC instead of TCP for the underlying transport protocol. Like HTTP/2, it does not obsolete previous major versions of the protocol. Support for HTTP/3 was added to Cloudflare and Google Chrome first,[16][17] and is also enabled in Firefox.[18] HTTP/3 has lower latency for real-world web pages, if enabled on the server, and loads faster than with HTTP/2, in some cases over three times faster than HTTP/1.1 (which is still commonly only enabled).[19] HTTP functions as a request–response protocol in the client–server model. A web browser, for example, may be the client whereas a process, named web server, running on a computer hosting one or more websites may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as HTML files and other content or performs other functions on behalf of the client, returns a response message to the client. The response contains completion status information about the request and may also contain requested content in its message body. A web browser is an example of a user agent (UA). Other types of user agent include the indexing software used by search providers (web crawlers), voice browsers, mobile apps, and other software that accesses, consumes, or displays web content. HTTP is designed to permit intermediate network elements to improve or enable communications between clients and servers. High-traffic websites often benefit from web cache servers that deliver content on behalf of upstream servers to improve response time. Web browsers cache previously accessed web resources and reuse them, whenever possible, to reduce network traffic. HTTP proxy servers at private network boundaries can facilitate communication for clients without a globally routable address, by relaying messages with external servers. To allow intermediate HTTP nodes (proxy servers, web caches, etc.) to accomplish their functions, some of the HTTP headers (found in HTTP requests/responses) are managed hop-by-hop whereas other HTTP headers are managed end-to-end (managed only by the source client and by the target web server). HTTP is an application layer protocol designed within the framework of the Internet protocol suite. Its definition presumes an underlying and reliable transport layer protocol.[20] In HTTP/3, the Transmission Control Protocol (TCP) is no longer used, but the older versions are still more used and they most commonly use TCP. They have also been adapted to use unreliable protocols such as the User Datagram Protocol (UDP), which HTTP/3 also (indirectly) always builds on, for example in HTTPU and Simple Service Discovery Protocol (SSDP). HTTP resources are identified and located on the network by Uniform Resource Locators (URLs), using the Uniform Resource Identifiers (URIs) schemes http and https. As defined in RFC 3986, URIs are encoded as hyperlinks in HTML documents, so as to form interlinked hypertext documents. In HTTP/1.0 a separate TCP connection to the same server is made for every resource request.[21] In HTTP/1.1 instead a TCP connection can be reused to make multiple resource requests (i.e. of HTML pages, frames, images, scripts, stylesheets, etc.).[22][23] HTTP/1.1 communications therefore experience less latency as the establishment of TCP connections presents considerable overhead, especially under high traffic conditions.[24] HTTP/2 is a revision of previous HTTP/1.1 in order to maintain the same client–server model and the same protocol methods but with these differences in order: to use a compressed binary representation of metadata (HTTP headers) instead of a textual one, so that headers require much less space; to use a single TCP/IP (usually encrypted) connection per accessed server domain instead of 2 to 8 TCP/IP connections; to use one or more bidirectional streams per TCP/IP connection in which HTTP requests and responses are broken down and transmitted in small packets to almost solve the problem of the HOLB (head-of-line blocking).[note 1] to add a push capability to allow server application to send data to clients whenever new data is available (without forcing clients to request periodically new data to server by using polling methods).[25] HTTP/2 communications therefore experience much less latency and, in most cases, even higher speeds than HTTP/1.1 communications. HTTP/3 is a revision of previous HTTP/2 in order to use QUIC + UDP transport protocols instead of TCP. Before that version, TCP/IP connections were used; but now, only the IP layer is used (which UDP, like TCP, builds on). This slightly improves the average speed of communications and to avoid the occasional (very rare) problem of TCP connection congestion that can temporarily block or slow down the data flow of all its streams (another form of "head of line blocking"). Tim Berners-Lee The term hypertext was coined by Ted Nelson in 1965 in the Xanadu Project, which was in turn inspired by Vannevar Bush's 1930s vision of the microfilm-based information retrieval and management "memex" system described in his 1945 essay "As We May Think". Tim Berners-Lee and his team at CERN are credited with inventing the original HTTP, along with HTML and the associated technology for a web server and a client user interface called web browser. Berners-Lee designed HTTP in order to help with the adoption of his other idea: the "WorldWideWeb" project, which was first proposed in 1989, now known as the World Wide Web. The first web server went live in 1990.[26][27] The protocol used had only one method, namely GET, which would request a page from a server.[28] The response from the server was always an HTML page.[2] Version Year introduced Current status Usage in August 2024[update] Support in August 2024[update] HTTP/0.9 1991 Obsolete 0 100% HTTP/1.0 1996 Obsolete 0 100% HTTP/1.1 1997 Standard 33.8% 100% HTTP/2 2015 Standard 35.3% 66.2% HTTP/3 2022 Standard 30.9% 30.9% In 1991, the first documented official version of HTTP was written as a plain document, less than 700 words long, and this version was named HTTP/0.9, which supported only GET method, allowing clients to only retrieve HTML documents from the server, but not supporting any other file formats or information upload.[2] Since 1992, a new document was written to specify the evolution of the basic protocol towards its next full version. It supported both the simple request method of the 0.9 version and the full GET request that included the client HTTP version. This was the first of the many unofficial HTTP/1.0 drafts that preceded the final work on HTTP/1.0.[3] After having decided that new features of HTTP protocol were required and that they had to be fully documented as official RFCs, in early 1995 the HTTP Working Group (HTTP WG, led by Dave Raggett) was constituted with the aim to standardize and expand the protocol with extended operations, extended negotiation, richer meta-information, tied with a security protocol which became more efficient by adding additional methods and header fields.[29][30] The HTTP WG planned to revise and publish new versions of the protocol as HTTP/1.0 and HTTP/1.1 within 1995, but, because of the many revisions, that timeline lasted much more than one year.[31] The HTTP WG planned also to specify a far future version of HTTP called HTTP-NG (HTTP Next Generation) that would have solved all remaining problems, of previous versions, related to performances, low latency responses, etc. but this work started only a few years later and it was never completed. In May 1996, RFC 1945 was published as a final HTTP/1.0 revision of what had been used in previous 4 years as a pre-standard HTTP/1.0-draft which was already used by many web browsers and web servers. In early 1996 developers started to even include unofficial extensions of the HTTP/1.0 protocol (i.e. keep-alive connections, etc.) into their products by using drafts of the upcoming HTTP/1.1 specifications.[32] Since early 1996, major web browsers and web server developers also started to implement new features specified by pre-standard HTTP/1.1 drafts specifications. End-user adoption of the new versions of browsers and servers was rapid. In March 1996, one web hosting company reported that over 40% of browsers in use on the Internet used the new HTTP/1.1 header "Host" to enable virtual hosting, and that by June 1996, 65% of all browsers accessing their servers were pre-standard HTTP/1.1 compliant.[33] In January 1997, RFC 2068 was officially released as HTTP/1.1 specifications. In June 1999, RFC 2616 was released to include all improvements and updates based on previous (obsolete) HTTP/1.1 specifications. Resuming the old 1995 plan of previous HTTP Working Group, in 1997 an HTTP-NG Working Group was formed to develop a new HTTP protocol named HTTP-NG (HTTP New Generation). A few proposals / drafts were produced for the new protocol to use multiplexing of HTTP transactions inside a single TCP/IP connection, but in 1999, the group stopped its activity passing the technical problems to IETF.[34] In 2007, the IETF HTTP Working Group (HTTP WG bis or HTTPbis) was restarted firstly to revise and clarify previous HTTP/1.1 specifications and secondly to write and refine future HTTP/2 specifications (named httpbis).[35][36] In 2009, Google, a private company, announced that it had developed and tested a new HTTP binary protocol named SPDY. The implicit aim was to greatly speed up web traffic (specially between future web browsers and its servers). SPDY was indeed much faster than HTTP/1.1 in many tests and so it was quickly adopted by Chromium and then by other major web browsers.[37] Some of the ideas about multiplexing HTTP streams over a single TCP/IP connection were taken from various sources, including the work of W3C HTTP-NG Working Group. In January–March 2012, HTTP Working Group (HTTPbis) announced the need to start to focus on a new HTTP/2 protocol (while finishing the revision of HTTP/1.1 specifications), maybe taking in consideration ideas and work done for SPDY.[38][39] After a few months about what to do to develop a new version of HTTP, it was decided to derive it from SPDY.[40] In May 2015, HTTP/2 was published as RFC 7540 and quickly adopted by all web browsers already supporting SPDY and more slowly by web servers. In June 2014, the HTTP Working Group released an updated six-part HTTP/1.1 specification obsoleting RFC 2616: RFC 7230, HTTP/1.1: Message Syntax and Routing RFC 7231, HTTP/1.1: Semantics and Content RFC 7232, HTTP/1.1: Conditional Requests RFC 7233, HTTP/1.1: Range Requests RFC 7234, HTTP/1.1: Caching RFC 7235, HTTP/1.1: Authentication In RFC 7230 Appendix-A, HTTP/0.9 was deprecated for servers supporting HTTP/1.1 version (and higher):[41]Since HTTP/0.9 did not support header fields in a request, there is no mechanism for it to support name-based virtual hosts (selection of resource by inspection of the Host header field). Any server that implements name-based virtual hosts ought to disable support for HTTP/0.9. Most requests that appear to be HTTP/0.9 are, in fact, badly constructed HTTP/1.x requests caused by a client failing to properly encode the request-target. Since 2016 many product managers and developers of user agents (browsers, etc.) and web servers have begun planning to gradually deprecate and dismiss support for HTTP/0.9 protocol, mainly for the following reasons:[42] it is so simple that an RFC document was never written (there is only the original document);[2] it has no HTTP headers and lacks many other features that nowadays are required for minimal security reasons; it has not been widespread since 1999..2000 (because of HTTP/1.0 and HTTP/1.1) and is commonly used only by some very old network hardware, i.e. routers, etc. [note 2] In 2020, the first drafts HTTP/3 were published and major web browsers and web servers started to adopt it. On 6 June 2022, IETF standardized HTTP/3 as RFC 9114.[43] In June 2022, a batch of RFCs was published, deprecating many of the previous documents and introducing a few minor changes and a refactoring of HTTP semantics description into a separate document. RFC 9110, HTTP Semantics RFC 9111, HTTP Caching RFC 9112, HTTP/1.1 RFC 9113, HTTP/2 RFC 9114, HTTP/3 (see also the section above) RFC 9204, QPACK: Field Compression for HTTP/3 RFC 9218, Extensible Prioritization Scheme for HTTP HTTP is a stateless application-level protocol and it requires a reliable network transport connection to exchange data between client and server.[20] In HTTP implementations, TCP/IP connections are used using well-known ports (typically port 80 if the connection is unencrypted or port 443 if the connection is encrypted, see also List of TCP and UDP port numbers).[44][45] In HTTP/2, a TCP/IP connection plus multiple protocol channels are used. In HTTP/3, the application transport protocol QUIC over UDP is used. Data is exchanged through a sequence of request–response messages which are exchanged by a session layer transport connection.[20] An HTTP client initially tries to connect to a server establishing a connection (real or virtual). An HTTP(S) server listening on that port accepts the connection and then waits for a client's request message. The client sends its HTTP request message. Upon receiving the request the server sends back an HTTP response message, which includes header(s) plus a body if it is required. The body of this response message is typically the requested resource, although an error message or other information may also be returned. At any time (for many reasons) client or server can close the connection. Closing a connection is usually advertised in advance by using one or more HTTP headers in the last request/response message sent to server or client.[22] Main article: HTTP persistent connection In HTTP/0.9, the TCP/IP connection is always closed after server response has been sent, so it is never persistent. In HTTP/1.0, as stated in RFC 1945, the TCP/IP connection should always be closed by server after a response has been sent.[note 3] In HTTP/1.1 a keep-alive-mechanism was officially introduced so that a connection could be reused for more than one request/response. Such persistent connections reduce request latency perceptibly because the client does not need to re-negotiate the TCP 3-Way-Handshake connection after the first request has been sent. Another positive side effect is that, in general, the connection becomes faster with time due to TCP's slow-start-mechanism. HTTP/1.1 added also HTTP pipelining in order to further reduce lag time when using persistent connections by allowing clients to send multiple requests before waiting for each response. This optimization was never considered really safe because a few web servers and many proxy servers, specially transparent proxy servers placed in Internet / Intranets between clients and servers, did not handle pipelined requests properly (they served only the first request discarding the others, they closed the connection because they saw more data after the first request or some proxies even returned responses out of order etc.). Because of this, only HEAD and some GET requests (i.e. limited to real file requests and so with URLs without query string used as a command, etc.) could be pipelined in a safe and idempotent mode. After many years of struggling with the problems introduced by enabling pipelining, this feature was first disabled and then removed from most browsers also because of the announced adoption of HTTP/2. HTTP/2 extended the usage of persistent connections by multiplexing many concurrent requests/responses through a single TCP/IP connection. HTTP/3 does not use TCP/IP connections but QUIC + UDP (see also: technical overview). HTTP/0.9 A requested resource was always sent in its entirety. HTTP/1.0 HTTP/1.0 added headers to manage resources cached by client in order to allow conditional GET requests; in practice a server has to return the entire content of the requested resource only if its last modified time is not known by client or if it changed since last full response to GET request. One of these headers, "Content-Encoding", was added to specify whether the returned content of a resource was or was not compressed. If the total length of the content of a resource was not known in advance (i.e. because it was dynamically generated, etc.) then the header "Content-Length: number" was not present in HTTP headers and the client assumed that when server closed the connection, the content had been sent in its entirety. This mechanism could not distinguish between a resource transfer successfully completed and an interrupted one (because of a server / network error or something else). HTTP/1.1 HTTP/1.1 introduced: new headers to better manage the conditional retrieval of cached resources. chunked transfer encoding to allow content to be streamed in chunks in order to reliably send it even when the server does not know its length in advance (i.e. because it is dynamically generated, etc.). byte range serving, where a client can request only one or more portions (ranges of bytes) of a resource (i.e. the first part, a part in the middle or in the end of the entire content, etc.) and the server usually sends only the requested part(s). This is useful to resume an interrupted download (when a file is very large), when only a part of a content has to be shown or dynamically added to the already visible part by a browser (i.e. only the first or the following n comments of a web page) in order to spare time, bandwidth and system resources, etc. HTTP/2, HTTP/3 Both HTTP/2 and HTTP/3 have kept the above mentioned features of HTTP/1.1. HTTP provides multiple authentication schemes such as basic access authentication and digest access authentication which operate via a challenge–response mechanism whereby the server identifies and issues a challenge before serving the requested content. HTTP provides a general framework for access control and authentication, via an extensible set of challenge–response authentication schemes, which can be used by a server to challenge a client request and by a client to provide authentication information.[1] The authentication mechanisms described above belong to the HTTP protocol and are managed by client and server HTTP software (if configured to require authentication before allowing client access to one or more web resources), and not by the web applications using a web application session. The HTTP Authentication specification also provides an arbitrary, implementation-specific construct for further dividing resources common to a given root URI. The realm value string, if present, is combined with the canonical root URI to form the protection space component of the challenge. This in effect allows the server to define separate authentication scopes under one root URI.[1] HTTP is a stateless protocol. A stateless protocol does not require the web server to retain information or status about each user for the duration of multiple requests. Some web applications need to manage user sessions, so they implement states, or server side sessions, using for instance HTTP cookies[46] or hidden variables within web forms. To start an application user session, an interactive authentication via web application login must be performed. To stop a user session a logout operation must be requested by user. These kind of operations do not use HTTP authentication but a custom managed web application authentication. Request messages are sent by a client to a target server.[note 4] A client sends request messages to the server, which consist of:[47] a request line, consisting of the case-sensitive request method, a space, the requested URI, another space, the protocol version, a carriage return, and a line feed, e.g.: GET /images/logo.png HTTP/1.1 zero or more request header fields (at least 1 or more headers in case of HTTP/1.1), each consisting of the case-insensitive field name, a colon, optional leading whitespace, the field value, an optional trailing whitespace and ending with a carriage return and a line feed, e.g.: Host: www.example.com Accept-Language: en an empty line, consisting of a carriage return and a line feed; an optional message body. In the HTTP/1.1 protocol, all header fields except Host: hostname are optional. A request line containing only the path name is accepted by servers to maintain compatibility with HTTP clients before the HTTP/1.0 specification in RFC 1945.[48] An HTTP/1.1 request made using telnet. The request message, response header section, and response body are highlighted. HTTP defines methods (sometimes referred to as verbs, but nowhere in the specification does it mention verb) to indicate the desired action to be performed on the identified resource. What this resource represents, whether pre-existing data or data that is generated dynamically, depends on the implementation of the server. Often, the resource corresponds to a file or the output of an executable residing on the server. The HTTP/1.0 specification[49] defined the GET, HEAD, and POST methods as well as listing the PUT, DELETE, LINK and UNLINK methods under additional methods. However, the HTTP/1.1 specification[50] formally defined and added five new methods: PUT, DELETE, CONNECT, OPTIONS, and TRACE. Any client can use any method and the server can be configured to support any combination of methods. If a method is unknown to an intermediate, it will be treated as an unsafe and non-idempotent method. There is no limit to the number of methods that can be defined, which allows for future methods to be specified without breaking existing infrastructure. For example, WebDAV defined seven new methods and RFC 5789 specified the PATCH method. Method names are case sensitive.[51][52] This is in contrast to HTTP header field names which are case-insensitive.[53] GET The GET method requests that the target resource transfer a representation of its state. GET requests should only retrieve data and should have no other effect. (This is also true of some other HTTP methods.)[1] For retrieving resources without making changes, GET is preferred over POST, as they can be addressed through a URL. This enables bookmarking and sharing and makes GET responses eligible for caching, which can save bandwidth. The W3C has published guidance principles on this distinction, saying, "Web application design should be informed by the above principles, but also by the relevant limitations."[54] See safe methods below. HEAD The HEAD method requests that the target resource transfer a representation of its state, as for a GET request, but without the representation data enclosed in the response body. This is useful for retrieving the representation metadata in the response header, without having to transfer the entire representation. Uses include checking whether a page is available through the status code and quickly finding the size of a file (Content-Length). POST The POST method requests that the target resource process the representation enclosed in the request according to the semantics of the target resource. For example, it is used for posting a message to an Internet forum, subscribing to a mailing list, or completing an online shopping transaction.[55] PUT The PUT method requests that the target resource create or update its state with the state defined by the representation enclosed in the request. A distinction from POST is that the client specifies the target location on the server.[56] DELETE The DELETE method requests that the target resource delete its state. CONNECT The CONNECT method requests that the intermediary establish a TCP/IP tunnel to the origin server identified by the request target. It is often used to secure connections through one or more HTTP proxies with TLS.[57][58] See HTTP CONNECT method. OPTIONS The OPTIONS method requests that the target resource transfer the HTTP methods that it supports. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource. TRACE The TRACE method requests that the target resource transfer the received request in the response body. That way a client can see what (if any) changes or additions have been made by intermediaries. PATCH The PATCH method requests that the target resource modify its state according to the partial update defined in the representation enclosed in the request. This can save bandwidth by updating a part of a file or document without having to transfer it entirely.[59] All general-purpose web servers are required to implement at least the GET and HEAD methods, and all other methods are considered optional by the specification.[52] Properties of request methods Request method RFC Request has payload body Response has payload body Safe Idempotent Cacheable GET RFC 9110 Optional Yes Yes Yes Yes HEAD RFC 9110 Optional No Yes Yes Yes POST RFC 9110 Yes Yes No No Yes PUT RFC 9110 Yes Yes No Yes No DELETE RFC 9110 Optional Yes No Yes No CONNECT RFC 9110 Optional Yes No No No OPTIONS RFC 9110 Optional Yes Yes Yes No TRACE RFC 9110 No Yes Yes Yes No PATCH RFC 5789 Yes Yes No No No A request method is safe if a request with that method has no intended effect on the server. The methods GET, HEAD, OPTIONS, and TRACE are defined as safe. In other words, safe methods are intended to be read-only. Safe methods can still have side effects not seen by the client, such as appending request information to a log file or charging an advertising account. In contrast, the methods POST, PUT, DELETE, CONNECT, and PATCH are not safe. They may modify the state of the server or have other effects such as sending an email. Such methods are therefore not usually used by conforming web robots or web crawlers; some that do not conform tend to make requests without regard to context or consequences. Despite the prescribed safety of GET requests, in practice their handling by the server is not technically limited in any way. Careless or deliberately irregular programming can allow GET requests to cause non-trivial changes on the server. This is discouraged because of the problems which can occur when web caching, search engines, and other automated agents make unintended changes on the server. For example, a website might allow deletion of a resource through a URL such as which, if arbitrarily fetched, even using GET, would simply delete the article.[60] A properly coded website would require a DELETE or POST method for this action, which non-malicious bots would not make. One example of this occurring in practice was during the short-lived Google Web Accelerator beta, which prefetched arbitrary URLs on the page a user was viewing, causing records to be automatically altered or deleted en masse. The beta was suspended only weeks after its first release, following widespread criticism.[61][60] See also: Idempotence § Computer science meaning A request method is idempotent if multiple identical requests with that method have the same effect as a single such request. The methods PUT and DELETE, and safe methods are defined as idempotent. Safe methods are trivially idempotent, since they are intended to have no effect on the server whatsoever; the PUT and DELETE methods, meanwhile, are idempotent since successive identical requests will be ignored. A website might, for instance, set up a PUT endpoint to modify a user's recorded email address. If this endpoint is configured correctly, any requests which ask to change a user's email address to the same email address which is already recorded—e.g. duplicate requests following a successful request—will have no effect. Similarly, a request to DELETE a certain user will have no effect if that user has already been deleted. In contrast, the methods POST, CONNECT, and PATCH are not necessarily idempotent, and therefore sending an identical POST request multiple times may further modify the state of the server or have further effects, such as sending multiple emails. In some cases this is the desired effect, but in other cases it may occur accidentally. A user might, for example, inadvertently send multiple POST requests by clicking a button again if they were not given clear feedback that the first click was being processed. While web browsers may show alert dialog boxes to warn users in some cases where reloading a page may re-submit a POST request, it is generally up to the web application to handle cases where a POST request should not be submitted more than once. Note that whether or not a method is idempotent is not enforced by the protocol or web server. It is perfectly possible to write a web application in which (for example) a database insert or other non-idempotent action is triggered by a GET or other request. To do so against recommendations, however, may result in undesirable consequences, if a user agent assumes that repeating the same request is safe when it is not. See also: Web cache A request method is cacheable if responses to requests with that method may be stored for future reuse. The methods GET, HEAD, and POST are defined as cacheable. In contrast, the methods PUT, DELETE, CONNECT, OPTIONS, TRACE, and PATCH are not cacheable. See also: List of HTTP header fields § Request fields Request header fields allow the client to pass additional information beyond the request line, acting as request modifiers (similarly to the parameters of a procedure). They give information about the client, about the target resource, or about the expected handling of the request. A response message is sent by a server to a client as a reply to its former request message.[note 4] A server sends response messages to the client, which consist of:[47] a status line, consisting of the protocol version, a space, the response status code, another space, a possibly empty reason phrase, a carriage return and a line feed, e.g.: HTTP/1.1 200 OK zero or more response header fields, each consisting of the case-insensitive field name, a colon, optional leading whitespace, the field value, an optional trailing whitespace and ending with a carriage return and a line feed, e.g.: Content-Type: text/html an empty line, consisting of a carriage return and a line feed; an optional message body. See also: List of HTTP status codes In HTTP/1.0 and since, the first line of the HTTP response is called the status line and includes a numeric status code (such as "404") and a textual reason phrase (such as "Not Found"). The response status code is a three-digit integer code representing the result of the server's attempt to understand and satisfy the client's corresponding request. The way the client handles the response depends primarily on the status code, and secondarily on the other response header fields. Clients may not understand all registered status codes but they must understand their class (given by the first digit of the status code) and treat an unrecognized status code as being equivalent to the x00 status code of that class. The standard reason phrases are only recommendations, and can be replaced with "local equivalents" at the web developer's discretion. If the status code indicated a problem, the user agent might display the reason phrase to the user to provide further information about the nature of the problem. The standard also allows the user agent to attempt to interpret the reason phrase, though this might be unwise since the standard explicitly specifies that status codes are machine-readable and reason phrases are human-readable. The first digit of the status code defines its class: 1XX (informational) The request was received, continuing process. 2XX (successful) The request was successfully received, understood, and accepted. 3XX (redirection) Further action needs to be taken in order to complete the request. 4XX (client error) The request contains bad syntax or cannot be fulfilled. 5XX (server error) The server failed to fulfill an apparently valid request. See also: List of HTTP header fields § Response fields The response header fields allow the server to pass additional information beyond the status line, acting as response modifiers. They give information about the server or about further access to the target resource or related resources. Each response header field has a defined meaning which can be further refined by the semantics of the request method or response status code. Below is a sample HTTP transaction between an HTTP/1.1 client and an HTTP/1.1 server running on www.example.com, port 80.[note 5][note 6] GET / HTTP/1.1 Host: www.example.com User-Agent: Mozilla/5.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8 Accept-Language: en-GB,en;q=0.5 Accept-Encoding: gzip, deflate, br Connection: keep-alive A client request (consisting in this case of the request line and a few headers that can be reduced to only the "Host: hostname" header) is followed by a blank line, so that the request ends with a double end of line, each in the form of a carriage return followed by a line feed. The "Host: hostname" header value distinguishes between various DNS names sharing a single IP address, allowing name-based virtual hosting. While optional in HTTP/1.0, it is mandatory in HTTP/1.1. (A "/" (slash) will usually fetch a /index.html file if there is one.) HTTP/1.1 200 OK Date: Mon, 23 May 2005 22:38:34 GMT Content-Type: text/html; charset=UTF-8 Content-Length: 155 Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux) ETag: "3f80f-1b6-3e1cb03b" Accept-Ranges: bytes Connection: close An Example Page Hello World, this is a very simple HTML document. The ETag (entity tag) header field is used to determine if a cached version of the requested resource is identical to the current version of the resource on the server. "Content-Type" specifies the Internet media type of the data conveyed by the HTTP message, while "Content-Length" indicates its length in bytes. The HTTP/1.1 webserver publishes its ability to respond to requests for certain byte ranges of the document by setting the field "Accept-Ranges: bytes". This is useful, if the client needs to have only certain portions[62] of a resource sent by the server, which is called byte serving. When "Connection: close" is sent, it means that the web server will close the TCP connection immediately after the end of the transfer of this response.[22] Most of the header lines are optional but some are mandatory. When header "Content-Length: number" is missing in a response with an entity body then this should be considered an error in HTTP/1.0 but it may not be an error in HTTP/1.1 if header "Transfer-Encoding: chunked" is present. Chunked transfer encoding uses a chunk size of 0 to mark the end of the content. Some old implementations of HTTP/1.0 omitted the header "Content-Length" when the length of the body entity was not known at the beginning of the response and so the transfer of data to client continued until server closed the socket. A "Content-Encoding: gzip" can be used to inform the client that the body entity part of the transmitted data is compressed by gzip algorithm. The most popular way of establishing an encrypted HTTP connection is HTTPS.[63] Two other methods for establishing an encrypted HTTP connection also exist: Secure Hypertext Transfer Protocol, and using the HTTP/1.1 Upgrade header to specify an upgrade to TLS. Browser support for these two is, however, nearly non-existent.[64][65][66] The Gopher protocol is a content delivery protocol that was displaced by HTTP in the early 1990s. The SPDY protocol is an alternative to HTTP developed at Google, superseded by HTTP/2. The Gemini protocol is a Gopher-inspired protocol which mandates privacy-related features. HTTP Persistence Compression HTTPS QUIC Request methods OPTIONS GET HEAD POST PUT DELETE TRACE CONNECT PATCH Header fields Cookie ETag Location HTTP referer DNT X-Forwarded-For Response status codes 301 Moved Permanently 302 Found 303 See Other 403 Forbidden 404 Not Found 451 Unavailable for Legal Reasons Security access control methods Basic access authentication Digest access authentication Security vulnerabilities HTTP header injection HTTP request smuggling HTTP response splitting HTTP parameter pollution vte InterPlanetary File System – can replace HTTP Comparison of file transfer protocols Constrained Application Protocol – a semantically similar protocol to HTTP but used UDP or UDP-like messages targeted for devices with limited processing capability; re-uses HTTP and other internet concepts like Internet media type and web linking (RFC 5988)[67] Content negotiation Digest access authentication HTTP compression HTTP/2 – developed by the IETF's Hypertext Transfer Protocol (httpbis) working group[36] List of HTTP header fields List of HTTP status codes Representational state transfer (REST) Variant object Wireless Application Protocol Web cache WebSocket ^ In practice, these streams are used as multiple TCP/IP sub-connections to multiplex concurrent requests/responses, thus greatly reducing the number of real TCP/IP connections on server side, from 2..8 per client to 1, and allowing many more clients to be served at once. ^ In 2022, HTTP/0.9 support has not been officially completely deprecated and is still present in many web servers and browsers (for server responses only), even if usually disabled. It is unclear how long it will take to decommission HTTP/0.9. ^ Since late 1996, some developers of popular HTTP/1.0 browsers and servers (specially those who had planned support for HTTP/1.1 too), started to deploy (as an unofficial extension) a sort of keep-alive-mechanism (by using new HTTP headers) in order to keep the TCP/IP connection open for more than a request/response pair and so to speed up the exchange of multiple requests/responses.[32] ^ a b HTTP/2 and HTTP/3 have a different representation for HTTP methods and headers. ^ HTTP/1.0 has the same messages except for a few missing headers. ^ HTTP/2 and HTTP/3 use the same request / response mechanism but with different representations for HTTP headers. ^ a b c d Fielding, R.; Nottingham, M.; Reschke, J. (June 2022). HTTP Semantics. IETF. doi:10.17487/RFC9110. RFC 9110. ^ a b c d Tim Berner-Lee (1991-01-01). "The Original HTTP as defined in 1991". www.w3.org. World Wide Web Consortium. Retrieved 2010-07-24. ^ a b Tim Berner-Lee (1992). "Basic HTTP as defined in 1992". www.w3.org. World Wide Web Consortium. Retrieved 2021-10-19. ^ In RFC 1945. That specification was then overcome by HTTP/1.1. ^ RFC 2068 (1997) was obsoleted by RFC 2616 in 1999, which was obsoleted by RFC 7230 in 2014, which was obsoleted by RFC 9110 in 2022. ^ "Usage Statistics of Default protocol https for websites". w3techs.com. Retrieved 2024-01-05. ^ "Usage Statistics of HTTP/2 for websites". w3techs.com. Retrieved 2024-01-05. ^ "Usage Statistics of HTTP/3 for Websites, August 2024". w3techs.com. Retrieved 2024-08-13. ^ "Can I use... Support tables for HTML5, CSS3, etc". caniuse.com. Retrieved 2024-01-05. ^ Friedl, S.; Popov, A.; Langley, A.; Stephan, E. (July 2014). Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension. IETF. doi:10.17487/RFC7301. RFC 7301. ^ Belshe, M.; Peon, R.; Thomson, M. "Hypertext Transfer Protocol Version 2, Use of TLS Features". Archived from the original on 2013-07-15. Retrieved 2015-02-10. ^ Benjamin, David. Using TLS 1.3 with HTTP/2. doi:10.17487/RFC8740. RFC 8740. Retrieved 2020-06-02. This lowers the barrier for deploying TLS 1.3, a major security improvement over TLS 1.2. ^ HTTP/3. 6 June 2022. doi:10.17487/RFC9114. RFC 9114. Retrieved 2022-06-06. ^ "Usage Statistics of HTTP/3 for websites". w3techs.com. Retrieved 2024-01-08. ^ "Can I use... Support tables for HTML5, CSS3, etc". canIuse.com. Retrieved 2024-01-08. ^ Cimpanu, Catalin (26 September 2019). "Cloudflare, Google Chrome, and Firefox add HTTP/3 support". ZDNet. Retrieved 27 September 2019. ^ "HTTP/3: the past, the present, and the future". The Cloudflare Blog. 2019-09-26. Retrieved 2019-10-30. ^ "Firefox Nightly supports HTTP 3 – General – Cloudflare Community". 2019-11-19. Retrieved 2020-01-23. ^ "HTTP/3 is Fast". Request Metrics. Retrieved 2022-07-01. ^ a b c "Connections, Clients, and Servers". RFC 9110, HTTP Semantics. sec. 3.3. doi:10.17487/RFC9110. RFC 9110. ^ "Overall Operation". RFC 1945. pp. 6–8. sec. 1.3. doi:10.17487/RFC1945. RFC 1945. ^ a b c "Connection Management: Establishment". RFC 9112, HTTP/1.1. sec. 9.1. doi:10.17487/RFC9112. RFC 9112. ^ "Connection Management: Persistence". RFC 9112, HTTP/1.1. sec. 9.3. doi:10.17487/RFC9112. RFC 9112. ^ "Classic HTTP Documents". W3.org. 1998-05-14. Retrieved 2010-08-01. ^ "HTTP/2 Protocol Overview". RFC 9113, HTTP/2). sec. 2. doi:10.17487/RFC7540. RFC 7540. ^ "Invention Of The Web, Web History, Who Invented the Web, Tim Berners-Lee, Robert Cailliau, CERN, First Web Server". LivingInternet. Retrieved 2021-08-11. ^ Berners-Lee, Tim (1990-10-02). "daemon.c - TCP/IP based server for HyperText". www.w3.org. Retrieved 2021-08-11. ^ Berners-Lee, Tim. "HyperText Transfer Protocol". World Wide Web Consortium. Retrieved 31 August 2010. ^ Raggett, Dave. "Dave Raggett's Bio". World Wide Web Consortium. Retrieved 11 June 2010. ^ Raggett, Dave; Berners-Lee, Tim. "Hypertext Transfer Protocol Working Group". World Wide Web Consortium. Retrieved 29 September 2010. ^ Raggett, Dave. "HTTP WG Plans". World Wide Web Consortium. Retrieved 29 September 2010. ^ a b David Gourley; Brian Totty; Marjorie Sayer; Anshu Aggarwal; Sailu Reddy (2002). HTTP: The Definitive Guide. (excerpt of chapter: "Persistent Connections"). O'Reilly Media, inc. ISBN 9781565925090. Retrieved 2021-10-18. ^ "HTTP 1.1 Compliant Browsers". webcom.com. Archived from the original on 1998-02-04. Retrieved 2009-05-29. ^ "HTTP-NG Working Group". www.w3.org. World Wide Web Consortium. 1997. Retrieved 2021-10-19. ^ Web Administrator (2007). "HTTP Working Group". httpwg.org. IETF. Retrieved 2021-10-19. ^ a b Web Administrator (2007). "HTTP Working Group: charter httpbis". datatracker.ietf.org. IETF. Retrieved 2021-10-19. ^ "SPDY: An experimental protocol for a faster web". dev.chromium.org. Google. 2009-11-01. Retrieved 2021-10-19. ^ "Rechartering httpbis". IETF; HTTP WG. 2012-01-24. Retrieved 2021-10-19. ^ IESG Secretary (2012-03-19). "WG Action: RECHARTER: Hypertext Transfer Protocol Bis (httpbis)". IETF; HTTP WG. Retrieved 2021-10-19. ^ Ilya Grigorik; Surma (2019-09-03). "High Performance Browser Networking: Introduction to HTTP/2". developers.google.com. Google Inc. Retrieved 2021-10-19. ^ "Appendix-A: HTTP Version History". RFC 7230, HTTP/1.1: Message Syntax and Routing. p. 78. sec. A. doi:10.17487/RFC7230. RFC 7230. ^ Matt Menke (2016-06-30). "Intent to Deprecate and Remove: HTTP/0.9 Support". groups.google.com. Retrieved 2021-10-15. ^ HTTP/3. 6 June 2022. doi:10.17487/RFC9114. RFC 9114. Retrieved 2022-06-06. ^ "http URI Scheme". RFC 9110, HTTP Semantics. sec. 4.2.1. doi:10.17487/RFC9110. RFC 9110. ^ "https URI Scheme". RFC 9110, HTTP Semantics. sec. 4.2.2. doi:10.17487/RFC9110. RFC 9110. ^ Lee, Wei-Bin; Chen, Hsing-Bai; Chang, Shun-Shyan; Chen, Tzung-Her (2019-01-25). "Secure and efficient protection for HTTP cookies with self-verification". International Journal of Communication Systems. 32 (2): e3857. doi:10.1002/dac.3857. S2CID 59524143. ^ a b "Message format". RFC 9112: HTTP/1.1. sec. 2.1. doi:10.17487/RFC9112. RFC 9112. ^ "Apache Week. HTTP/1.1". Archived from the original on 2021-06-02. Retrieved 2021-05-03. 090502 apacheweek.com ^ Berners-Lee, Tim; Fielding, Roy T.; Nielsen, Henrik Frystyk. "Method Definitions". Hypertext Transfer Protocol – HTTP/1.0. IETF. pp. 30–32. sec. 8. doi:10.17487/RFC1945. RFC 1945. ^ "Method Definitions". RFC 2616. pp. 51–57. sec. 9. doi:10.17487/RFC2616. RFC 2616. ^ "Request Line". RFC 9112, HTTP/1.1. sec. 3. doi:10.17487/RFC9112. RFC 9112. ^ a b "Methods: Overview". RFC 9110, HTTP Semantics. sec. 9.1. doi:10.17487/RFC9110. RFC 9110. ^ "Header Fields". RFC 9110, HTTP Semantics. sec. 6.3. doi:10.17487/RFC9110. RFC 9110. ^ Jacobs, Ian (2004). "URIs, Addressability, and the use of HTTP GET and POST". Technical Architecture Group finding. W3C. Retrieved 26 September 2010. ^ "POST". RFC 9110, HTTP Semantics. sec. 9.3.3. doi:10.17487/RFC9110. RFC 9110. ^ "PUT". RFC 9110, HTTP Semantics. sec. 9.3.4. doi:10.17487/RFC9110. RFC 9110. ^ "CONNECT". RFC 9110, HTTP Semantics. sec. 9.3.6. doi:10.17487/RFC9110. RFC 9110. ^ "Vulnerability Note VU#150227: HTTP proxy default configurations allow arbitrary TCP connections". US-CERT. 2002-05-17. Retrieved 2007-05-10. ^ Dusseault, Lisa; Snell, James M. (March 2010). PATCH Method for HTTP. IETF. doi:10.17487/RFC5789. RFC 5789. ^ a b Ediger, Brad (2007-12-21). Advanced Rails: Building Industrial-Strength Web Apps in Record Time. O'Reilly Media, Inc. p. 188. ISBN 978-0596519728. A common mistake is to use GET for an action that updates a resource. [...] This problem came into the Rails public eye in 2005, when the Google Web Accelerator was released. ^ Cantrell, Christian (2005-06-01). "What Have We Learned From the Google Web Accelerator?". Adobe Blogs. Adobe. Archived from the original on 2017-08-19. Retrieved 2018-11-19. ^ Luotonen, Ari; Franks, John (February 22, 1996). Byte Range Retrieval Extension to HTTP. IETF. I-D draft-ietf-http-range-retrieval-00. ^ Canavan, John (2001). Fundamentals of Networking Security. Norwood, MA: Artech House. pp. 82–83. ISBN 9781580531764. ^ Zalewski, Michal. "Browser Security Handbook". Retrieved 30 April 2015. ^ "Chromium Issue 4527: implement RFC 2817: Upgrading to TLS Within HTTP/1.1". Retrieved 30 April 2015. ^ "Mozilla Bug 276813 – [RFE] Support RFC 2817 / TLS Upgrade for HTTP 1.1". Retrieved 30 April 2015. ^ Nottingham, Mark (October 2010). Web Linking. IETF. doi:10.17487/RFC5988. RFC 5988. Wikibooks has a book on the topic of: Communication Networks/HTTP Protocol Wikimedia Commons has media related to Hypertext Transfer Protocol. Official website IETF HTTP Working Group on GitHub "Change History for HTTP". W3.org. Retrieved 2010-08-01. A detailed technical history of HTTP. "Design Issues for HTTP". W3.org. Retrieved 2010-08-01. Design Issues by Berners-Lee when he was designing the protocol. Retrieved from "