**HTTP là gì?**

Đầu tiên, HTTP là gì? HTTP là một giao thức giao tiếp tầng ứng dụng dựa trên TCP/IP chuẩn hóa cách máy khách và máy chủ giao tiếp với nhau. Nó xác định cách nội dung được yêu cầu và truyền qua internet. Theo giao thức lớp ứng dụng, ý tôi là nó chỉ đơn giản là một lớp trừu tượng tiêu chuẩn hóa cách máy chủ (máy khách và máy chủ) giao tiếp. Bản thân HTTP phụ thuộc vào TCP/IP để nhận các yêu cầu và phản hồi giữa máy khách và máy chủ. Theo mặc định, cổng TCP 80 được sử dụng, nhưng cũng có thể sử dụng các cổng khác. Tuy nhiên, HTTPS sử dụng cổng 443.

**HTTP/0.9 - The One Liner (1991)**

Phiên bản đầu tiên được ghi chép lại của HTTP là HTTP/0.9 được đưa ra vào năm 1991. Đây là giao thức đơn giản nhất từng có; có một phương thức gọi là GET. Nếu một khách hàng phải truy cập một số trang web trên máy chủ, thì nó sẽ thực hiện một yêu cầu đơn giản như bên dưới:

GET /index.html

Và phản hồi từ máy chủ sẽ như sau:

(response body)

(connection closed)

Nghĩa là, máy chủ sẽ nhận yêu cầu, trả lời bằng HTML để phản hồi và ngay sau khi nội dung được truyền xong, kết nối sẽ bị đóng. đã có

* không có tiêu đề
* GET là phương pháp duy nhất được phép
* Phản hồi phải là HTML

Như bạn có thể thấy, giao thức thực sự không có gì khác hơn là một bước đệm cho những gì sắp tới.

**HTTP/1.0 - 1996**

Vào năm 1996, phiên bản tiếp theo của HTTP tức là HTTP/1.0 đã phát triển và cải thiện rất nhiều so với phiên bản gốc.

Không giống như HTTP/0.9 chỉ được thiết kế cho phản hồi HTML, HTTP/1.0 giờ đây có thể xử lý các định dạng phản hồi khác, chẳng hạn như hình ảnh, tệp video, văn bản thuần túy hoặc bất kỳ loại nội dung nào khác. Nó đã thêm nhiều phương thức hơn (tức là POST và HEAD), định dạng yêu cầu/phản hồi đã được thay đổi, tiêu đề HTTP đã được thêm vào cả yêu cầu và phản hồi, mã trạng thái đã được thêm vào để xác định phản hồi, hỗ trợ bộ ký tự đã được giới thiệu, loại nhiều phần, ủy quyền , bộ nhớ đệm, mã hóa nội dung và hơn thế nữa đã được đưa vào.

Đây là cách một yêu cầu và phản hồi HTTP/1.0 mẫu có thể trông như thế nào:

*GET / HTTP/1.0*

*Host: cs.fyi*

*User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_10\_5)*

*Accept: \*/\**

Như bạn có thể thấy, bên cạnh yêu cầu, ứng dụng khách cũng đã gửi thông tin cá nhân, loại phản hồi được yêu cầu, v.v. Trong khi ở HTTP/0.9, ứng dụng khách không bao giờ có thể gửi thông tin đó vì không có tiêu đề.

Phản hồi ví dụ cho yêu cầu ở trên có thể giống như bên dưới:

*HTTP/1.0 200 OK*

*Content-Type: text/plain*

*Content-Length: 137582*

*Expires: Thu, 05 Dec 1997 16:00:00 GMT*

*Last-Modified: Wed, 5 August 1996 15:55:28 GMT*

*Server: Apache 0.84*

*(response body)*

*(connection closed)*

Ở phần đầu của phản hồi có HTTP/1.0 (HTTP theo sau là số phiên bản), sau đó có mã trạng thái 200 theo sau là cụm từ lý do (hoặc mô tả mã trạng thái, nếu bạn muốn).

Trong phiên bản mới hơn này, các tiêu đề yêu cầu và phản hồi vẫn được giữ dưới dạng mã hóa ASCII, nhưng nội dung phản hồi có thể thuộc bất kỳ loại nào, ví dụ như hình ảnh, video, HTML, văn bản thuần túy hoặc bất kỳ loại nội dung nào khác. Vì vậy, bây giờ máy chủ đó có thể gửi bất kỳ loại nội dung nào tới máy khách; không lâu sau khi được giới thiệu, thuật ngữ "Siêu văn bản" trong HTTP đã trở thành cách gọi sai. Giao thức truyền tải HMTP hoặc Hypermedia có thể có ý nghĩa hơn, nhưng tôi đoán là chúng ta bị mắc kẹt với cái tên suốt đời.

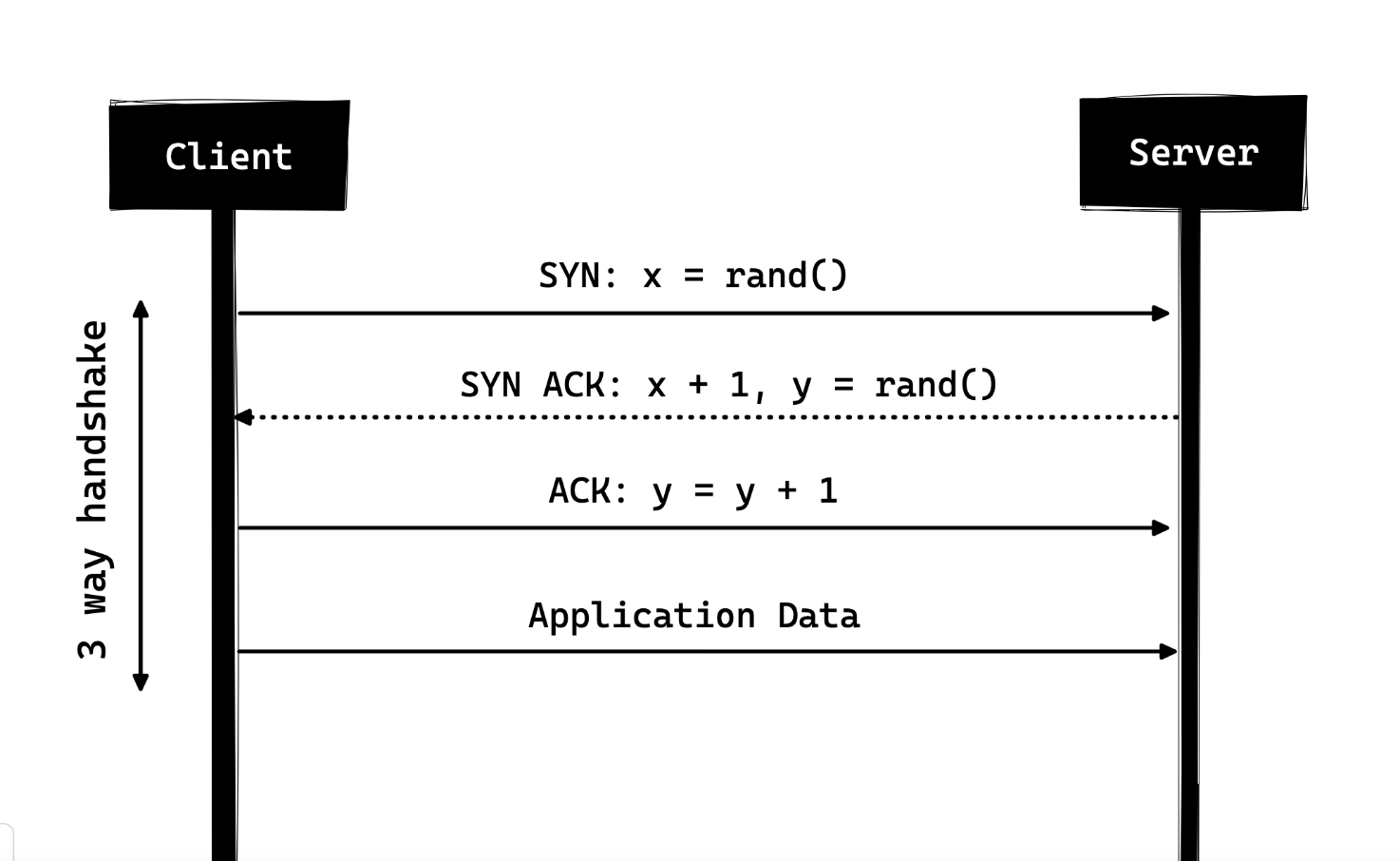
Một trong những nhược điểm chính của HTTP/1.0 là bạn không thể có nhiều yêu cầu trên mỗi kết nối. Nghĩa là, bất cứ khi nào máy khách cần thứ gì đó từ máy chủ, nó sẽ phải mở một kết nối TCP mới và sau khi yêu cầu đơn lẻ đó được đáp ứng, kết nối sẽ bị đóng. Và đối với bất kỳ yêu cầu tiếp theo nào, nó sẽ phải trên một kết nối mới. Tại sao nó xấu? Chà, giả sử rằng bạn truy cập một trang web có 10 hình ảnh, 5 biểu định kiểu và 5 tệp javascript, tổng cộng có 20 mục cần tìm nạp khi có yêu cầu tới trang web đó. Vì máy chủ đóng kết nối ngay sau khi yêu cầu được thực hiện nên sẽ có một loạt 20 kết nối riêng biệt trong đó từng mục sẽ được phục vụ từng mục một trên các kết nối riêng biệt của chúng.

**Bắt tay ba bước**

Bắt tay ba bước ở dạng đơn giản là tất cả các kết nối TCP bắt đầu bằng bắt tay ba bước trong đó máy khách và máy chủ chia sẻ một loạt gói trước khi bắt đầu chia sẻ dữ liệu ứng dụng.

* SYN - Máy khách chọn một số ngẫu nhiên, chẳng hạn như x và gửi số đó đến máy chủ.
* SYN ACK - Máy chủ xác nhận yêu cầu bằng cách gửi lại gói ACK cho máy khách bao gồm một số ngẫu nhiên, giả sử y được máy chủ chọn và số x+1 trong đó x là số do máy khách gửi
* ACK - Máy khách tăng số y nhận được từ máy chủ và gửi lại gói ACK với số y+1

Sau khi quá trình bắt tay ba bước hoàn tất, quá trình chia sẻ dữ liệu giữa máy khách và máy chủ có thể bắt đầu. Cần lưu ý rằng máy khách có thể bắt đầu gửi dữ liệu ứng dụng ngay khi nó gửi gói ACK cuối cùng nhưng máy chủ sẽ vẫn phải đợi gói ACK được nhận để thực hiện yêu cầu.



Tuy nhiên, một số triển khai của HTTP/1.0 đã cố gắng khắc phục sự cố này bằng cách giới thiệu một tiêu đề mới có tên là Kết nối: giữ nguyên để thông báo cho máy chủ "Này máy chủ, đừng đóng kết nối này, tôi cần lại". Tuy nhiên, nó vẫn không được hỗ trợ rộng rãi và vấn đề vẫn tiếp diễn.

Ngoài việc không kết nối, HTTP còn là một giao thức không trạng thái, tức là máy chủ không duy trì thông tin về máy khách và do đó, mỗi yêu cầu phải có thông tin cần thiết để máy chủ tự thực hiện yêu cầu mà không có bất kỳ liên kết nào với bất kỳ thông tin cũ nào. yêu cầu. Và do đó, điều này càng đổ thêm dầu vào lửa, tức là ngoài số lượng lớn kết nối mà máy khách phải mở, nó còn phải gửi một số dữ liệu dư thừa trên dây dẫn đến việc sử dụng băng thông tăng lên.

**HTTP/1.1 – 1997**

Chỉ sau 3 năm của HTTP/1.0, phiên bản tiếp theo tức là HTTP/1.1 đã được phát hành vào năm 1999; đã tạo ra rất nhiều cải tiến so với người tiền nhiệm của nó. Bao gồm các cải tiến chính so với HTTP/1.0

* Các phương thức HTTP mới đã được thêm vào, giới thiệu PUT, PATCH, OPTIONS, DELETE
* Nhận dạng tên máy chủ trong HTTP/1.0 Tiêu đề máy chủ không bắt buộc nhưng HTTP/1.1 đã yêu cầu nó.
* Kết nối liên tục Như đã thảo luận ở trên, trong HTTP/1.0 chỉ có một yêu cầu cho mỗi kết nối và kết nối đã bị đóng ngay sau khi yêu cầu được đáp ứng, điều này dẫn đến các vấn đề về độ trễ và hiệu suất nghiêm trọng. HTTP/1.1 đã giới thiệu các kết nối liên tục, tức là các kết nối không bị đóng theo mặc định và được giữ mở, cho phép nhiều yêu cầu tuần tự. Để đóng các kết nối, tiêu đề Kết nối: đóng phải có sẵn theo yêu cầu. Khách hàng thường gửi tiêu đề này trong yêu cầu cuối cùng để đóng kết nối một cách an toàn.
* Pipelining Nó cũng giới thiệu sự hỗ trợ cho pipelining, trong đó máy khách có thể gửi nhiều yêu cầu đến máy chủ mà không cần đợi phản hồi từ máy chủ trên cùng một kết nối và máy chủ phải gửi phản hồi theo cùng một trình tự mà các yêu cầu đã nhận được. Nhưng làm thế nào để khách hàng biết rằng đây là thời điểm mà quá trình tải xuống phản hồi đầu tiên hoàn tất và nội dung cho phản hồi tiếp theo bắt đầu, bạn có thể hỏi! Chà, để giải quyết vấn đề này, phải có tiêu đề Độ dài nội dung mà khách hàng có thể sử dụng để xác định nơi phản hồi kết thúc và nó có thể bắt đầu chờ phản hồi tiếp theo.

*Cần lưu ý rằng để hưởng lợi từ các kết nối liên tục hoặc đường dẫn, tiêu đề Độ dài nội dung phải có sẵn trên phản hồi, bởi vì điều này sẽ cho khách hàng biết khi quá trình truyền hoàn tất và nó có thể gửi yêu cầu tiếp theo (theo cách tuần tự thông thường của gửi yêu cầu) hoặc bắt đầu chờ phản hồi tiếp theo (khi tính năng đường ống dẫn được bật).*

*Nhưng vẫn còn một vấn đề với cách tiếp cận này. Và đó là, nếu dữ liệu là động và máy chủ không thể tìm thấy độ dài nội dung trước đó thì sao? Trong trường hợp đó, bạn thực sự không thể hưởng lợi từ các kết nối liên tục, phải không?! Để giải quyết vấn đề HTTP/1.1 đã giới thiệu mã hóa chunked này. Trong những trường hợp như vậy, máy chủ có thể bỏ qua Độ dài nội dung để chuyển sang mã hóa chunked (nhiều hơn nữa trong giây lát). Tuy nhiên, nếu không có sẵn thì kết nối phải được đóng lại khi kết thúc yêu cầu.*

* Truyền theo đoạn Trong trường hợp nội dung động, khi máy chủ thực sự không thể tìm ra Độ dài nội dung khi quá trình truyền bắt đầu, nó có thể bắt đầu gửi nội dung theo từng đoạn (từng đoạn) và thêm Độ dài nội dung cho từng đoạn khi nó được gửi . Và khi tất cả các khối được gửi, tức là toàn bộ quá trình truyền đã hoàn tất, nó sẽ gửi một đoạn trống, tức là đoạn có Độ dài nội dung được đặt thành 0 để xác định máy khách mà quá trình truyền đã hoàn tất. Để thông báo cho khách hàng về việc truyền chunked, máy chủ bao gồm tiêu đề Transfer-Encoding: chunked
* Không giống như HTTP/1.0 chỉ có xác thực Cơ bản, HTTP/1.1 bao gồm xác thực ủy quyền và thông báo
* Bộ nhớ đệm
* Phạm vi byte
* Bộ nhân vật
* đàm phán ngôn ngữ
* cookie của khách hàng
* Hỗ trợ nén nâng cao
* Mã trạng thái mới
* ..và hơn thế nữa

Tôi sẽ không nói sâu về tất cả các tính năng của HTTP/1.1 trong bài đăng này vì bản thân nó là một chủ đề và bạn có thể tìm thấy rất nhiều điều về nó. Một tài liệu như vậy mà tôi khuyên bạn nên đọc là Sự khác biệt chính giữa HTTP/1.0 và HTTP/1.1 và đây là liên kết tới RFC gốc dành cho những người đạt thành tích cao.

HTTP/1.1 được giới thiệu vào năm 1999 và nó đã trở thành tiêu chuẩn trong nhiều năm. Mặc dù, nó đã cải thiện rất nhiều so với người tiền nhiệm của nó; với việc web thay đổi hàng ngày, nó bắt đầu cho thấy tuổi của nó. Việc tải một trang web ngày nay tốn nhiều tài nguyên hơn bao giờ hết. Một trang web đơn giản ngày nay phải mở hơn 30 kết nối. Vâng, HTTP/1.1 có các kết nối liên tục, vậy tại sao lại có nhiều kết nối như vậy? bạn nói! Lý do là, trong HTTP/1.1, nó chỉ có thể có một kết nối nổi bật tại bất kỳ thời điểm nào. HTTP/1.1 đã cố gắng khắc phục điều này bằng cách giới thiệu đường ống nhưng nó không giải quyết được hoàn toàn vấn đề do chặn đầu dòng trong đó yêu cầu chậm hoặc nặng có thể chặn các yêu cầu phía sau và một khi yêu cầu bị kẹt trong đường ống, nó sẽ phải đợi các yêu cầu tiếp theo được thực hiện. Để khắc phục những thiếu sót này của HTTP/1.1,

**SPDY-2009**

Google đã đi trước và bắt đầu thử nghiệm các giao thức thay thế để làm cho web nhanh hơn và cải thiện bảo mật web đồng thời giảm độ trễ của các trang web. Năm 2009, họ công bố SPDY.

*SPDY là thương hiệu của Google và không phải là từ viết tắt.*

Người ta thấy rằng nếu chúng ta tiếp tục tăng băng thông, thì hiệu suất mạng sẽ tăng ngay từ đầu nhưng đến một lúc hiệu suất không tăng nhiều. Nhưng nếu bạn làm tương tự với độ trễ, tức là nếu chúng tôi tiếp tục giảm độ trễ, thì hiệu suất sẽ không đổi. Đây là ý tưởng cốt lõi để tăng hiệu suất đằng sau SPDY, giảm độ trễ để tăng hiệu suất mạng.

*Đối với những người không biết sự khác biệt, độ trễ là độ trễ tức là mất bao lâu để dữ liệu di chuyển giữa nguồn và đích (được đo bằng mili giây) và băng thông là lượng dữ liệu được truyền mỗi giây (bit mỗi giây).*

Các tính năng của SPDY bao gồm, ghép kênh, nén, ưu tiên, bảo mật, v.v. Tôi sẽ không đi sâu vào chi tiết của SPDY, vì bạn sẽ hiểu khi chúng ta tìm hiểu sâu về HTTP/2 trong phần tiếp theo khi tôi cho biết HTTP/2 hầu hết được lấy cảm hứng từ SPDY.

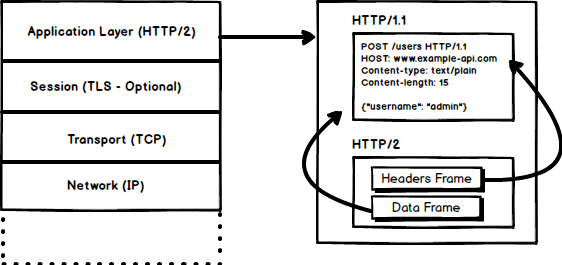
SPDY không thực sự cố gắng thay thế HTTP; đó là một lớp dịch qua HTTP tồn tại ở lớp ứng dụng và đã sửa đổi yêu cầu trước khi gửi nó đến dây. Nó bắt đầu trở thành một tiêu chuẩn không chính thức và phần lớn các trình duyệt bắt đầu triển khai nó.

Vào năm 2015, tại Google, họ không muốn có hai tiêu chuẩn cạnh tranh nên đã quyết định hợp nhất nó thành HTTP trong khi khai sinh ra HTTP/2 và loại bỏ SPDY.

**HTTP/2 – 2015**

Đến bây giờ, bạn phải tin rằng tại sao chúng tôi cần một bản sửa đổi khác của giao thức HTTP. HTTP/2 được thiết kế để truyền tải nội dung có độ trễ thấp. Các tính năng chính hoặc sự khác biệt so với phiên bản HTTP/1.1 cũ bao gồm:

* Nhị phân thay vì văn bản
* Ghép kênh - Nhiều yêu cầu HTTP không đồng bộ trên một kết nối
* Nén tiêu đề bằng HPACK
* Đẩy máy chủ - Nhiều phản hồi cho một yêu cầu
* Ưu tiên yêu cầu
* Bảo vệ



**1. Giao thức nhị phân**

HTTP/2 có xu hướng giải quyết vấn đề về độ trễ tăng lên tồn tại trong HTTP/1.x bằng cách biến nó thành một giao thức nhị phân. Là một giao thức nhị phân, nó dễ dàng phân tích cú pháp hơn nhưng không giống như HTTP/1.x, mắt người không thể đọc được nữa. Các khối xây dựng chính của HTTP/2 là Khung và Luồng

**Khung và Luồng**

Các thông báo HTTP hiện bao gồm một hoặc nhiều khung. Có khung HEADERS cho siêu dữ liệu và khung DATA cho tải trọng và tồn tại một số loại khung khác (HEADERS, DATA, RST\_STREAM, SETTINGS, PRIORITY, v.v.) mà bạn có thể kiểm tra thông qua thông số kỹ thuật HTTP/ 2 .

Mỗi yêu cầu và phản hồi HTTP/2 được cấp một ID luồng duy nhất và nó được chia thành các khung. Các khung không là gì ngoài các mẩu dữ liệu nhị phân. Một tập hợp các khung được gọi là Luồng. Mỗi khung có một id luồng xác định luồng mà nó thuộc về và mỗi khung có một tiêu đề chung. Ngoài ra, ngoài ID luồng là duy nhất, điều đáng nói là bất kỳ yêu cầu nào do khách hàng bắt đầu đều sử dụng số lẻ và phản hồi từ máy chủ có ID luồng số chẵn.

Ngoài các ĐẦU và DỮ LIỆU, một loại khung khác mà tôi nghĩ đáng đề cập ở đây là RST\_STREAM, đây là loại khung đặc biệt được sử dụng để hủy bỏ một số luồng, tức là máy khách có thể gửi khung này để cho máy chủ biết rằng tôi không cần luồng này nữa không. Trong HTTP/1.1, cách duy nhất để khiến máy chủ ngừng gửi phản hồi cho máy khách là đóng kết nối, điều này dẫn đến độ trễ tăng lên do phải mở một kết nối mới cho bất kỳ yêu cầu liên tiếp nào. Trong khi ở HTTP/2, máy khách có thể sử dụng RST\_STREAM và ngừng nhận một luồng cụ thể trong khi kết nối vẫn mở và các luồng khác vẫn đang phát.

**2. Ghép kênh**

Vì HTTP/2 hiện là một giao thức nhị phân và như tôi đã nói ở trên rằng nó sử dụng các khung và luồng cho các yêu cầu và phản hồi, nên khi kết nối TCP được mở, tất cả các luồng sẽ được gửi không đồng bộ qua cùng một kết nối mà không cần mở thêm bất kỳ kết nối nào. Và ngược lại, máy chủ phản hồi theo cùng một cách không đồng bộ, tức là phản hồi không có thứ tự và máy khách sử dụng id luồng được chỉ định để xác định luồng chứa một gói cụ thể. Điều này cũng giải quyết vấn đề chặn đầu dòng tồn tại trong HTTP/1.x, tức là máy khách sẽ không phải đợi yêu cầu mất nhiều thời gian và các yêu cầu khác sẽ vẫn được xử lý.

**3. Nén tiêu đề**

Nó là một phần của RFC riêng biệt nhằm mục đích tối ưu hóa các tiêu đề đã gửi. Bản chất của nó là khi chúng tôi liên tục truy cập máy chủ từ cùng một máy khách, sẽ có rất nhiều dữ liệu dư thừa mà chúng tôi gửi đi gửi lại trong các tiêu đề và đôi khi có thể có các cookie làm tăng kích thước tiêu đề dẫn đến việc sử dụng băng thông và tăng độ trễ. Để khắc phục điều này, HTTP/2 đã giới thiệu tính năng nén tiêu đề.

Không giống như yêu cầu và phản hồi, các tiêu đề không được nén ở định dạng gzip hoặc nén, v.v. nhưng có một cơ chế khác để nén tiêu đề, đó là các giá trị bằng chữ được mã hóa bằng mã Huffman và bảng tiêu đề được duy trì bởi máy khách và máy chủ và cả máy khách và máy chủ bỏ qua bất kỳ tiêu đề lặp lại nào (ví dụ: tác nhân người dùng, v.v.) trong các yêu cầu tiếp theo và tham chiếu chúng bằng cách sử dụng bảng tiêu đề do cả hai duy trì.

Trong khi chúng ta đang nói về các tiêu đề, hãy để tôi nói thêm ở đây rằng các tiêu đề vẫn giống như trong HTTP/1.1, ngoại trừ việc bổ sung một số tiêu đề giả, tức là `:method`, `:scheme` và :host``:path`

**4. Đẩy máy chủ**

Đẩy máy chủ là một tính năng tuyệt vời khác của HTTP/2 trong đó máy chủ, biết rằng máy khách sẽ yêu cầu một tài nguyên nhất định, có thể đẩy nó tới máy khách mà không cần máy khách yêu cầu. Ví dụ: giả sử một trình duyệt tải một trang web, nó phân tích cú pháp toàn bộ trang để tìm ra nội dung từ xa mà nó phải tải từ máy chủ, sau đó gửi các yêu cầu tiếp theo đến máy chủ để lấy nội dung đó.

Đẩy máy chủ cho phép máy chủ giảm các vòng lặp bằng cách đẩy dữ liệu mà nó biết rằng máy khách sẽ yêu cầu. Cách thực hiện là, máy chủ gửi một khung đặc biệt có tên PUSH\_PROMISE thông báo cho máy khách rằng "Này, tôi chuẩn bị gửi tài nguyên này cho bạn! Đừng hỏi tôi về nó." Khung PUSH\_PROMISE được liên kết với luồng khiến quá trình đẩy xảy ra và nó chứa ID luồng đã hứa, tức là luồng mà máy chủ sẽ gửi tài nguyên để được đẩy.

**5. Ưu tiên yêu cầu**

Máy khách có thể chỉ định mức độ ưu tiên cho một luồng bằng cách bao gồm thông tin về mức độ ưu tiên trong khung HEADERS mà luồng được mở. Vào bất kỳ thời điểm nào khác, khách hàng có thể gửi khung ƯU TIÊN để thay đổi mức độ ưu tiên của luồng.

Không có bất kỳ thông tin ưu tiên nào, máy chủ xử lý các yêu cầu không đồng bộ, tức là không có bất kỳ thứ tự nào. Nếu có mức độ ưu tiên được chỉ định cho một luồng, thì dựa trên thông tin về mức độ ưu tiên này, máy chủ sẽ quyết định lượng tài nguyên cần được cung cấp để xử lý yêu cầu nào.

**6. Bảo mật**

Đã có nhiều cuộc thảo luận về việc có nên bắt buộc bảo mật (thông qua TLS) đối với HTTP/2 hay không. Cuối cùng, nó đã được quyết định không bắt buộc. Tuy nhiên, hầu hết các nhà cung cấp đều tuyên bố rằng họ sẽ chỉ hỗ trợ HTTP/2 khi nó được sử dụng qua TLS. Vì vậy, mặc dù HTTP/2 không yêu cầu mã hóa theo thông số kỹ thuật nhưng dù sao thì nó cũng trở thành bắt buộc theo mặc định. Ngoài ra, HTTP/2 khi được triển khai qua TLS sẽ áp đặt một số yêu cầu, tức là. Phải sử dụng TLS phiên bản 1.2 trở lên, phải có một mức kích thước khóa tối thiểu nhất định, bắt buộc phải có khóa tạm thời, v.v.

**Tài liệu tham khảo:**

[**https://cs.fyi/guide/http-in-depth**](https://cs.fyi/guide/http-in-depth)