HTTP

HyperText Transfer Protocol

The HyperText Transfer Protocol, or HTTP, is a cornerstone of the modern Internet. Originally intended to transfer documents, it's now used for so much more, such as streaming media from companies like Netflix and applications through scripts that your browser downloads and runs. In this segment I'll explain the basic conceptual model behind HTTP and present some analytical tools for understanding how it performs.

HyperText

History [edit source | edit beta]

The term HyperText was coined by Ted Nelson who in turn was inspired by Vannevar Bush's microfilm-based "memex". Tim Berners-Lee first proposed the "WorldWideWeb" project — now known as the World Wide Web. Berners-Lee and his team are credited with inventing the original HTTP along with HTML and the associated technology for a web server and a text-based web browser. The first version of the protocol had only one method, namely GET, which would request a page from a server.[3] The response from the server was always an HTML page.[4]

Tim Berners-Lee

The first documented version of HTTP was HTTP V0.9 (1991). Dave Raggett led the HTTP Working Group (HTTP WG) in 1995 and wanted to 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.[5][6] RFC 1945 officially introduced and recognized HTTP V1.0 in 1996.

The HTTP WG planned to publish new standards in December 1995[7] and the support for pre-standard HTTP/1.1 based on the then developing RFC 2068 (called HTTP-NG) was rapidly adopted by the major browser developers in early 1996. By March 1996, pre-standard HTTP/1.1 was supported in Arena,[8] Netscape 2.0,[8] Netscape Navigator Gold 2.01,[8] Mosaic 2.7,[citation needed] Lynx 2.5[citation needed], and in Internet Explorer 2.0[citation needed]. End-user adoption of the new browsers was rapid. In March 1996, one web hosting company reported that over 40% of browsers in use on the Internet were HTTP 1.1 compliant.[citation needed] That same web hosting company reported that by June 1996, 65% of all browsers accessing their servers were HTTP/1.1 compliant.[9] The HTTP/1.1 standard as defined in RFC 2068 was officially released in January 1997. Improvements and updates to the HTTP/1.1 standard were released under RFC 2616 in June 1999.

```
218 <h2><span class="mw-headline" id="History">History</span><span class="mw-editsection"><span class="mw-editsection-bracket">[</span><a href="/w/index.p
219 <div class="thumb tright">
220 <div class="thumbinner" style="width:192px;"><a href="/wiki/File:Tim_Berners-Lee_CP_2.jpg" class="image"><img alt="" src="//upload.wikimedia.org/wikip
221 <div class="thumbcaption">
222 <div class="magnify"><a href="/wiki/File:Tim_Berners-Lee_CP_2.jpg" class="internal" title="Enlarge"><img src="//bits.wikimedia.org/static-1.22wmf15/sk
223 <a href="/wiki/Tim_Berners-Lee" title="Tim Berners-Lee">Tim Berners-Lee</a></div>
224 </div>
225 </div>
226 <p>The term <a href="/wiki/HyperText" title="HyperText" class="mw-redirect">HyperText</a> was coined by <a href="/wiki/Ted_Nelson" title="Ted Nelson">
227 <p>The first documented version of HTTP was <b><a rel="nofollow" class="external text" href="http://www.w3.org/pub/WWW/Protocols/HTTP/AsImplemented.ht
228 <p>The HTTP WG planned to publish new standards in December 1995<sup id="cite_ref-7" class="reference"><a href="#cite_note-7"><span>[</span>7<span>]</
```


HTTP stands for HyperText Transfer Protocol -- so what's HyperText? HyperText is a document format that lets you include formatting and content information in a document. Whenever you download a web page, you're downloading a hypertext document. Unlike many other document formats, like Microsoft Word or PDF, hypertext is all ASCII text. If you look at a document, there generally speaking aren't any characters your regular text editor can't display.

So let's take as an example this excerpt from the Wikipedia page on HTTP's history. It has the word "History" is a larger font, some links, shown as blue, an embedded image, and a few other nice bits of formatting, such as the line under history, that make it easier to read.

Under the covers, the document looks like this: this is the hypertext my browser downloaded to display this section. All of the formatting information is inside angle brackets. This less-than, H2, greater than, for example, means this is a heading, so should be displayed bigger. You can see the word "History" outside any such formatting information -- the word History on this snippet is displayed as a header, as you can see. So at a basic level, a hypertext document is just a text document, which your browser displays based on these special formatting controls, called tags. A hypertext link, for example, is just a formatting tag that says "the stuff inside this tag, if clicked, should load this URL." The tag that does this is the A tag -- see this example here of an A tag for the HTTP V0.9 link on line 227. When you click on that link, it takes you to this URL: http://www.w3.org/pub/" etc.

HyperText

History [edit source | edit beta]

The term HyperText was coined by Ted Nelson who in turn was inspired by Vannevar Bush's microfilm-based "memex". Tim Berners-Lee first proposed the "WorldWideWeb" project — now known as the World Wide Web. Berners-Lee and his team are credited with inventing the original HTTP along with HTML and the associated technology for a web server and a text-based web browser. The first version of the protocol had only one method, namely GET, which would request a page from a server.[3] The response from the server was always an HTML page.[4]


Tim Berners-Lee

The first documented version of HTTP was HTTP V0.9 (1991). Dave Raggett led the HTTP Working Group (HTTP WG) in 1995 and wanted to 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.[5][6] RFC 1945 officially introduced and recognized HTTP V1.0 in 1996.

The HTTP WG planned to publish new standards in December 1995[7] and the support for pre-standard HTTP/1.1 based on the then developing RFC 2068 (called HTTP-NG) was rapidly adopted by the major browser developers in early 1996. By March 1996, pre-standard HTTP/1.1 was supported in Arena,[8] Netscape 2.0,[8] Netscape Navigator Gold 2.01,[8] Mosaic 2.7,[citation needed] Lynx 2.5[citation needed], and in Internet Explorer 2.0[citation needed]. End-user adoption of the new browsers was rapid. In March 1996, one web hosting company reported that over 40% of browsers in use on the Internet were HTTP 1.1 compliant.[citation needed] That same web hosting company reported that by June 1996, 65% of all browsers accessing their servers were HTTP/1.1 compliant.[9] The HTTP/1.1 standard as defined in RFC 2068 was officially released in January 1997. Improvements and updates to the HTTP/1.1 standard were released under RFC 2616 in June 1999.

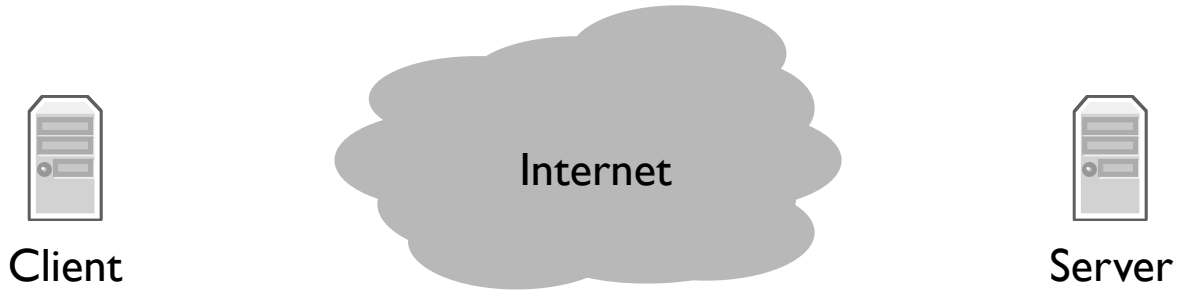
```
218 <h2><span class="mw-headline" id="History">History</span><span class="mw-editsection"><span class="mw-editsection-bracket">[</span><a href="/w/index.p
219 <div class="thumb tright">
220 <div class="thumbinner" style="width:192px;"><a href="/wiki/File:Tim_Berners-Lee_CP_2.jpg" class="image"><img alt="" src="//upload.wikimedia.org/wikip
221 <div class="thumbcaption">
222 <div class="magnify"><a href="/wiki/File:Tim_Berners-Lee_CP_2.jpg" class="internal" title="Enlarge"><img src="//bits.wikimedia.org/static-1.22wmf15/sk
223 <a href="/wiki/Tim_Berners-Lee" title="Tim Berners-Lee">Tim Berners-Lee</a></div>
224 </div>
225 </div>
226 <p>The term <a href="/wiki/HyperText" title="HyperText" class="mw-redirect">HyperText</a> was coined by <a href="/wiki/Ted_Nelson" title="Ted Nelson">
227 <p>The first documented version of HTTP was <b><a rel="nofollow" class="external text" href="http://www.w3.org/pub/WWW/Protocols/HTTP/AsImplemented.ht
228 <p>The HTTP WG planned to publish new standards in December 1995<sup id="cite_ref-7" class="reference"><a href="#cite_note-7"><span>[</span>7<span>]</
```


But there's one way in which a hypertext document is more than just formatting. With hypertext, you can embed documents, or files, inside other files. The simplest example of this on this wikipedia page is the image. The bits of the image aren't stored in this hypertext document. That wouldn't be human readable ASCII text. Instead, there's a way to, in a hypertext document, say "load this other document and put it here." Take a look at line 220 -- you'll see an image, or IMG tag. The IMG tag says "load the image from this URL and display it here." When your browser loads the hypertext for the wikipedia page, it sees tags like this one and automatically requests the files they reference. So when you load the page your browser automatically requests the image and displays it.

There are all kinds of resources besides images that a web page can reference: other pages, style sheets, scripts, fonts, and more. Let's look at an example: I'm going to request the web page for the New York Times and use my browser's developer tools to see all of the requests this results in. As you can see, it requests something on the order of 20 documents, ranging from hypertext to images to ads.

This turns out to be a really important property of hypertext: requesting one document can lead to to request more documents.

World Wide Web (HTTP)

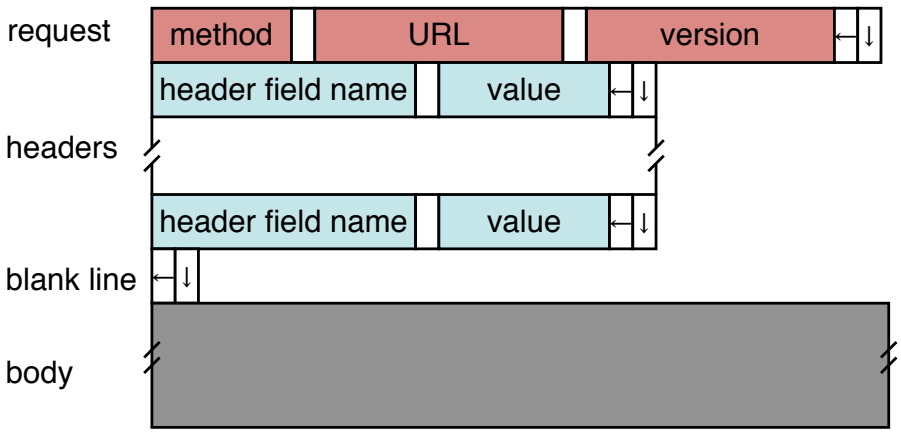


In HTTP, a client opens a TCP connection to a server and sends commands to it. The most common command is GET, which requests a page. HTTP was designed to be a document-centric way for programs to communicate. For example, if I type http://www.stanford.edu/ in my browser, the browser opens a connection to the server www.stanford.edu and sends a GET request for the root page of the site. The server receives the request, checks if it's valid and the user can access that page, and sends a response. The response has a numeric code associated with it. For example, if the server sends a 200 OK response to a GET, this means that the request was accepted and the rest of the response has the document data. In the example of the www.stanford.edu web page, a 200 OK response would include the HyperText that describes the main Stanford page. There are other kinds of requests, such as PUT, DELETE, and INFO, as well as other responses such as 400 Bad Request.

Like hypertext itself, HTTP is all in ASCII text: it's human readable. For example, the beginning of a GET request for the New York Times looks like this: GET / HTTP/1.1. The beginning of a response to a successful request looks like this: HTTP/1.1 200 OK.

But the basic model is simple: client sends a request by writing to the connection, the server reads the request, processes it, and writes a response to the connection, which the client then reads. The data the client reads might cause it to then issue more GET requests.

HTTP Request Format



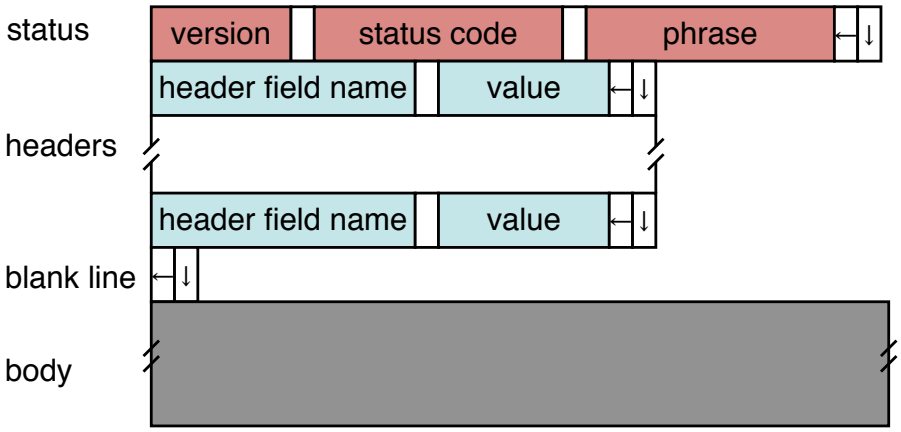
This is what an HTTP request looks like. The first line, ASCII text, says the method, such as GET, the URL for the method, and the version of HTTP being used. The white boxes represent spaces. So there's a space between method and URL as well as a space between URL and version. The left arrow means carriage return -- a way to say to go to the beginning of the line, and the down arrow means newline, a way to say to go to a new line. So, for example, in my prior example of requesting this URL, the method will be GET, the URL will be fullduplex/index.html, and the version will be most likely HTTP/1.1.

After this first line, the request itself, there's zero or more headers. There's one header per line. Each header line starts with the header field name, followed by the value.

After the headers, there's an empty line. Followed by the body of the message.

Wait -- why might a request have a body? What's the body of a request? In the case of the GET method, to request a page, the body is empty. But HTTP supports other methods, such as POST, which sends data, for example when you fill out a form and submit it. POST requests often have a body.

HTTP Response



An HTTP response looks similar. The first line has the HTTP version, the status code, and the phrase associated with that status code. Such as 200 OK or 404 Not Found. There's then zero or more headers, a blank line, and the body of the response.

Let's see what the response to my GET request looks like. It's a 304 -- this web page has not been modified since my browser put it in its cache. Now if I clear my browser cache, and request the page again, the request doesn't have a Modified-Since header and so the response is a 200 OK.

The developer tools on Firefox let you see the request/response pair but not their actual formats. For that, I'll do something much simpler. I'll use the telnet program to connect to a web server. Telnet opens a TCP connection: it writes what you type to the socket and prints out what it reads. So I'll telnet to sing.stanford.edu port 80 and type GET /fullduplex/index.html HTTP/1.0.

A lot of HTML comes back. If I scroll to the top, I can see the HTTP response, 200 OK, with a bunch of headers, a newline, then the body, the HTML of the page. The Content-Length header tells me how long the body is.

HTTP

- Cornerstone application protocol of modern Internet
- Text protocol, human readable
- Request/response API
- Document-centric

HTTP is a cornerstone protocol of the modern Internet. While it was originally document-centric, designed to fetch pages and documents, today it's used for much more. A document, for example, can be a script that your browser executes as part of an application. The basic model, however, of requesting URLs and receiving responses, still holds. One nice thing about HTTP is that it's human readable text. You can type an HTTP request and read the response, as you saw me do by telnetting to port 80. I encourage you to play around a bit, use the developer tools in your browser to see what's requested, and learn more about the details of the protocol.

HTTP/1.0

- Open connection
- Issue GET
- Server closes connection after response

So that's the basics of the protocol. Request, response. HTTP/1.0 was very simple. A client wanting to request a document opens a connection. It sends a GET request. The server responds with a status code, such as 200 OK, the document, and closes the connection once the response is complete. If the client wants to request a second document, it must open a second connection.

When the web was mostly text, with maybe an image or two, this approach worked just fine. People hand-wrote their web pages, putting in all of the formatting.

HTTP/1.0 speed

- Latency: 50ms
- Request size: 1 full segment
- Response size: 2 full segments (size of slow start window)
- Segment packetization delay: 10ms (request and response), full duplex
- Maximum open connections: 2, teardown is instantaneous

So let's walk through how long this takes. Let's make some simplifying assumptions. The latency between the client and server is 50 milliseconds. An HTTP request is a full TCP segment. A response is 2 full segments, so the size of a small initial slow start congestion window. That way we don't have to worry about window sizes, etc. The packetization delay of a full segment is 10ms. So the packetization delay of a request is 10 milliseconds and a reply is 20 milliseconds. Let's finally assume that TCP segments with no data, such as the three-way handshake and ACK packets, have a packetization delay of zero. Connection teardown is instantaneous. Finally, we can have up to 2 open connections.

HTTP/1.0 speed

- Latency: 50ms
- Request size: 1 full segment
- Response size: 2 full segments (size of slow start window)
- Segment packetization delay: 10ms (request and response), full duplex
- Maximum open connections: 2, teardown is instantaneous

- Case 1: Single page

So let's walk through how long this takes. Let's make some simplifying assumptions. The web server can respond immediately, there's no processing delay. The latency between the client and server is 50 milliseconds. An HTTP request is a full TCP segment. A response is 2 full segments, so the size of a small initial slow start congestion window. That way we don't have to worry about window sizes, etc. The packetization delay of a full segment is 10ms. So the total packetization delay of a request is 10 milliseconds and a reply is 20 milliseconds. You can assume that the links are full duplex, such that a node can simultaneously receive and transmit on the same link. This means the packetization delay of a request does not affect the packetization delay of a response. Let's finally assume that TCP segments with no data, such as the three-way handshake and ACK packets, have a packetization delay of zero. Finally, we can have up to 2 open connections.

Let's consider a first case. We want to load a single page. How long will this take?

First, there's the latency of sending a SYN, so 50ms. There's the latency of the SYN-ACK, so another 50ms. On receiving the SYN/ACK, the client can send the ACK of the three way handshake, followed by the request. The request has a packetization delay of 10ms, so this takes 60ms. The server then needs to send the response back. The packetization delay of the response is 20ms, so this step is 70ms.

HTTP/1.0 speed

- Latency: 50ms
- Request size: 1 full segment
- Response size: 2 full segments (size of slow start window)
- Segment packetization delay: 10ms (request and response), full duplex
- Maximum open connections: 2, teardown is instantaneous

- Case 1: Single page
  - SYN: 50ms, SYN/ACK: 50ms, ACK/request: 60ms, response: 70ms

So the total delay is 50 milliseconds plus 50 milliseconds plus 60 milliseconds plus 70 milliseconds, or 230 milliseconds.

HTTP/1.0 speed

- Latency: 50ms
- Request size: 1 full segment
- Response size: 2 full segments (size of slow start window)
- Segment packetization delay: 10ms (request and response), full duplex
- Maximum open connections: 2, teardown is instantaneous

- Case 1: Single page, 230ms
  - SYN: 50ms, SYN/ACK: 50ms, ACK/request: 60ms, response: 70ms

HTTP/1.0 speed

- Latency: 50ms
- Request size: 1 full segment
- Response size: 2 full segments (size of slow start window)
- Segment packetization delay: 10ms (request and response), full duplex
- Maximum open connections: 2, teardown is instantaneous

- Case 1: Single page, 230ms
  - SYN: 50ms, SYN/ACK: 50ms, ACK/request: 60ms, response: 70ms
- Case 2: Page that loads 2 images
  - Step 1 (page) - Setup: 100ms, request/response: 130ms
  - Step 2 (images) - 100ms, request/response: ?

Let's look at a more complex example. There's a page that loads 2 images.

We can break this into two steps. In the first step, the client requests the page. In the second step, it uses 2 connections to request the images.

The first step will take the same length as our single page example. There's 100 millisecond for the setup, then 130 milliseconds for the request and response.

The second step is a bit trickier. Remember, while we have separate TCP connections, they are sharing the same link. This means that the packetization delay of one request affects the other. Setting up the two connections will take 100 milliseconds. But how long will it take for the two request/responses to complete?

Quiz

- Latency: 50ms
- Request size: 1 full segment
- Response size: 2 full segments (size of slow start window)
- Segment packetization delay: 10ms (request and response), full duplex

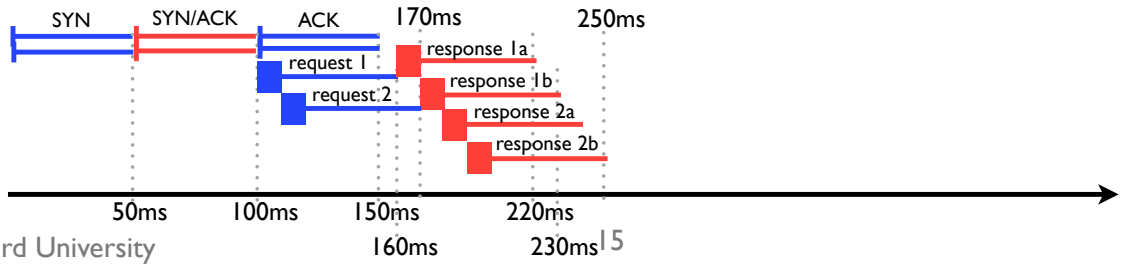
- Case 2: Page that loads 2 images
  - Step 1 (page) - Setup: 100ms, request/response: 130ms
  - Step 2 (images) - 100ms, request/response: ?

Here's the quiz. Given that we have two parallel requests sharing the same link, and the parameters above, how long will the request/response exchange take?

Quiz

- Latency: 50ms
- Request size: 1 full segment
- Response size: 2 full segments (size of slow start window)
- Segment packetization delay: 10ms (request and response), full duplex

- Case 2: Page that loads 2 images
  - Step 1 (page) - Setup: 100ms, request/response: 130ms
  - Step 2 (images) - 100ms, request/response: ?



Answer: It will take 150 milliseconds, so 250 milliseconds in total. In this figure, blue lines are segments from the client to server, and red are from the server to client. The SYN/SYNACK exchange takes 100 milliseconds. The first request takes 60 milliseconds to arrive, at which point, 160 milliseconds, the server can begin sending a response. It enqueues two segments to send. As the first response segment goes out over the link, the server receives the second request, and enqueues 2 more response segments. This means that the responses will take a total of 90 milliseconds to arrive after the first request arrives. The additional packetization delay of the 2nd request is masked by the queueing of responses.

HTTP/1.0 speed

- Latency: 50ms
- Request size: 1 full segment
- Response size: 2 full segments (size of slow start window)
- Segment packetization delay: 10ms (request and response), full duplex
- Maximum open connections: 2, teardown is instantaneous

- Case 1: Single page, 230ms
  - SYN: 50ms, SYN/ACK: 50ms, ACK/request: 60ms, response: 70ms
- Case 2: Page that loads 2 images: 480ms
  - Setup: 100ms, request/response: 130ms
  - Setup: 100ms, request/response: 150ms

Quiz

Given the following parameters, how long will HTTP 1.0 take to load these pages?

- Latency: 20ms
- Request size: 1 full segment
- Response size: 2 full segments, initial congestion window is 10 segments
- Segment packetization delay: 5ms (request and response)
- Maximum open connections: 2, teardown is instantaneous

- Case 1: Single page
- Case 2: Page that loads 5 images

Here's a quiz. Given the following parameters, how long will it take HTTP 1.0 to load these pages? For Case 2, think very carefully about how requests and responses might overlap.

Quiz

Given the following parameters, how long will HTTP 1.0 take to load these pages?

- Latency: 20ms
- Request size: 1 full segment
- Response size: 2 full segments, initial congestion window is 10 segments
- Segment packetization delay: 5ms (request and response)
- Maximum open connections: 2, teardown is instantaneous

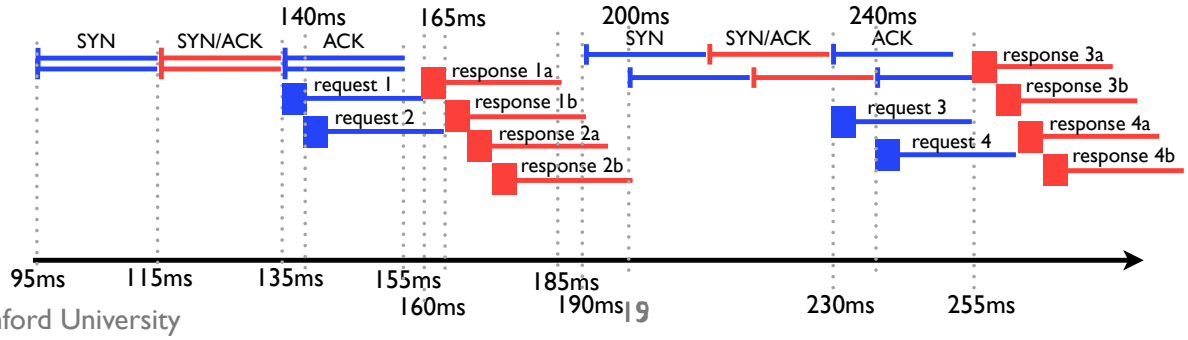
- Case 1: Single page, 95ms (20ms + 20ms + 25ms + 30ms)
- Case 2: Page that loads 5 images, 380ms (95ms + 95ms + 95ms + 95ms)

The answer to case 1 is 95ms. The setup is 40 milliseconds, the ACK/request is 25 milliseconds, and the response is 30 milliseconds. So 95 ms total.

The answer to case 2 is 380 millseconds. It takes 95 milliseconds to load the initial page. It then takes 95 milliseconds to load image 1. When image 1 finishes, image 3 starts. Meanwhile, image 2 is already in flight. So that's 95 milliseconds. When image 3 completes, that's another 95 milliseconds and image 2 has already completed. Image 4 is in flight. It takes a final 95 milliseconds for image 5. For a total of 380 milliseconds.

Case 2

- Latency: 20ms
- Request size: 1 full segment
- Response size: 2 full segments, initial congestion window is 10 segments
- Segment packetization delay: 5ms (request and response)
- Maximum open connections: 2, teardown is instantaneous
- Case 2: Page that loads 5 images, 380ms (95ms + 95ms + 95ms + 95ms)



Let's look at this pictorially to see what's happening. This picture starts after the first initial page request. It's showing what happens as the client requests images. So we start at 95 milliseconds. There's a pair of SYN/SYN-ACKs as the two connections start their three way handshake. So 40 milliseconds later, at 135 milliseconds, the client sends request 1, at 135 millseconds, then request 2, at 140 milliseconds.

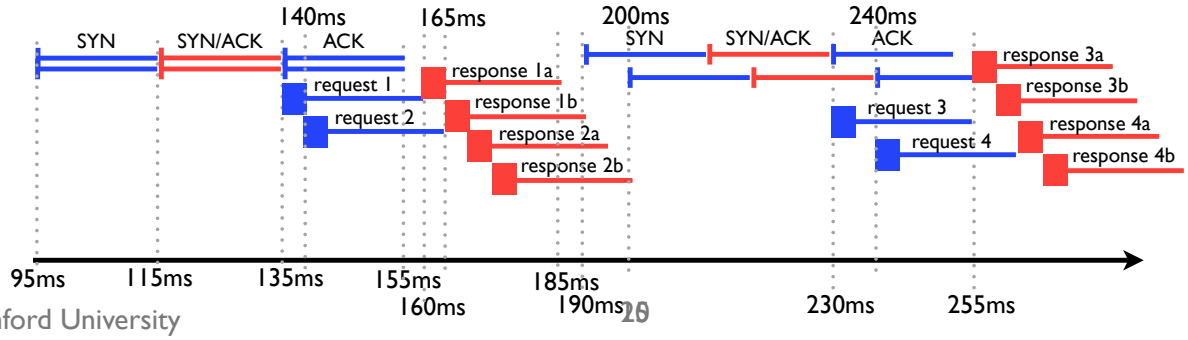
Request 1 arrives at the server at 165 milliseconds: 20 milliseconds of latency and 5 milliseconds of packetization delay. The server starts sending the response. It's sent one segment of the response, 1a, when the second request arrives. The response segments for the second request are enqueued and are sent after response 1b.

Response 1b arrives at the client at 190 milliseconds. At this point, the client opens a new connection, through a three-way handshake. But note how long this took. The client is requesting the third image at 190 milliseconds: 95 milliseconds after the first request started. Because the second request is going in parallel, the client doesn't have to wait for it to complete before starting the third request. It will start the fifth request immediately when the third one completes. So these three rounds take on 95 milliseconds each. If we'd requested 6 images, then the final round would take 105 milliseconds.

Look at this figure carefully until you understand what's going in. As requests are delayed going out from queueing, they delay the responses. As responses are delayed from going out due to queueing, they further delay requests. Over time, this causes the requests and responses to naturally space themselves out, reducing queueing delay. And because we have multiple operations in parallel, they can mask each other's latencies.

Case 2

- Latency: 20ms
- Request size: 1 full segment
- Response size: 2 full segments, initial congestion window is 10 segments
- Segment packetization delay: 5ms (request and response)
- Maximum open connections: 2, teardown is instantaneous
- Case 2: Page that loads 5 images, 380ms (95ms + 95ms + 95ms + 95ms)



If you look at these numbers and think about them a bit, you should see that requesting multiple resources in parallel doesn't take much longer than requesting a single resource. There's additional packetization delay, but in most networks today packetization delay is a tiny fraction of the overall time. A single request can't fill the network capacity, but many request might be able to. HTTP/1.0 only allows a single request per connection, though.