1. HTTP messages that travel between the client and the server. HTTP/1.1 and HTTP/2 both use this strategy, but there A common method of optimizing web applications is to use compression algorithms to reduce the size of are implementation problems in the former that prohibit compressing the entire message. The following section will discuss why this is the case, and how HTTP/2 can provide a solution.

**HTTP/1.1**

* Programs like [gzip](https://www.gzip.org/) have long been used to compress the data sent in HTTP messages, especially to decrease the size of CSS and JavaScript files.
* The header component of a message, however, is always sent as plain text. Although each header is quite small, the burden of this uncompressed data weighs heavier and heavier on the connection as more requests are made, particularly penalizing complicated, API-heavy web applications that require many different resources and thus many different resource requests.
* Additionally, the use of cookies can sometimes make headers much larger, increasing the need for some kind of compression.
* In order to solve this bottleneck, HTTP/2 uses HPACK compression to shrink the size of headers, a topic discussed further in the next section.

**HTTP/2**

* One of the themes that has come up again and again in HTTP/2 is its ability to use the binary framing layer to exhibit greater control over finer detail.
* The same is true when it comes to header compression.
* HTTP/2 can split headers from their data, resulting in a header frame and a data frame.
* The HTTP/2-specific compression program [HPACK](https://tools.ietf.org/html/draft-ietf-httpbis-header-compression-12) can then compress this header frame.
* This algorithm can encode the header metadata using Huffman coding, thereby greatly decreasing its size.
* Additionally, HPACK can keep track of previously conveyed metadata fields and further compress them according to a dynamically altered index shared between the client and the server.

