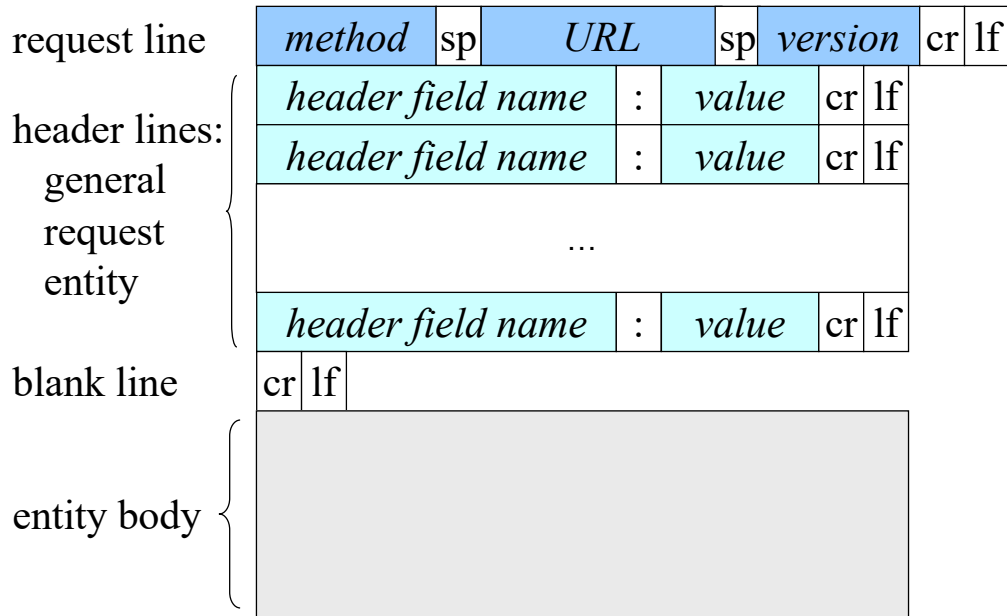


# http

## HyperText Transfer Protocol (HTTP)

- Lingua franca of the Web
- Application-layer protocol built on top of TCP
- Request-Response type of protocol
  - Request: e.g. “GET *URL HTTP\_VERSION*”
  - Response from server
- Requests and responses are encoded in text
- Stateless: sequence of requests and responses are independent
  - Done for scalability and simplicity at the server
  - Cookies maintain session state outside of HTTP

# General Request Format

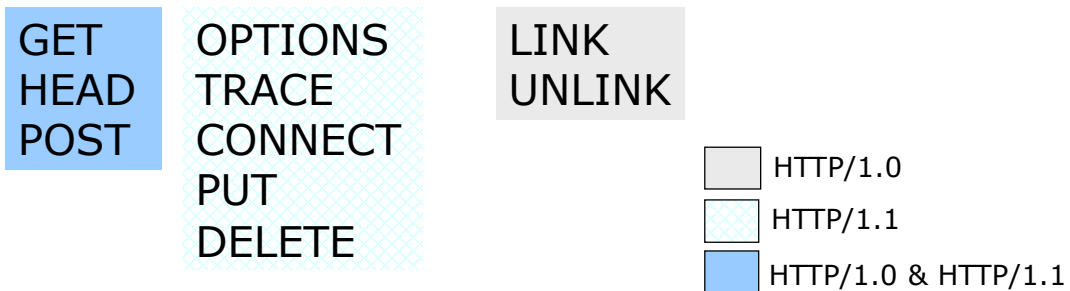


## GET Request (E.g.)

```
GET / HTTP/1.1
Date: Sat, 3 Mar 2007 10:00:02 GMT
Accept: image/gif,image/x-xbitmap,image/jpeg,*/
Accept-Encoding: gzip
Accept-Language: en-us
Pragma: no-cache
Host: www.school.edu
From: me@abc.edu
User-Agent: Mozilla/4.0
Cookie: A=abcedfeg; B=xyz
Connection: Keep-Alive
```

# Request Methods

- Request methods can have "idempotent" property
- The side-effects of  $N > 0$  identical requests is the same as for a single request
  - GET, HEAD, PUT and DELETE share this property
  - OPTIONS and TRACE should not have side effects, so are inherently idempotent



## Request Methods (cont' d)

- GET method available since HTTP 0.9
  - Uses '?' to pass arguments to the URI resource
  - Proxies and servers log requests
  - Often servers limit the length of the URI
- POST method implies the entity to be passed to the request URI for processing
  - Intended to allow transmitting larger data than in GET
  - It may not be idempotent (vs. GET)
  - Server has to determine the entity size; e.g., can use Content-Length

## Method: PUT

- PUT method indicates to server to accept entity body and store it, using the request URI
  - Either create a new resource (201 Created) or replace an existing one
  - Requires verifying Authorization header (else error)
  - Could qualify request with entity tag (e.g., If-Match header)

## Method: PUT

- With PUT, server must satisfy any Content-\* header, i.e., it's capable of decoding entity
  - Content-Type: application/postscript
  - Content-Encoding: gzip
- PUT is idempotent (i.e., no side effects)
  - Only the resource identified by URI would be changed

## Method: PATCH

- A newer method
- A PATCH request gives a set of instructions on how to modify a resource
  - Whereas PUT gives a complete representation of a resource
- A PATCH is not necessarily idempotent
  - PATCH (like POST) may have side-effects on other resources

## Method: OPTIONS

- OPTIONS: query for capabilities
- Determine the version of HTTP that the server supports
  - E.g., a proxy can verify that the server complies with a specific version of the protocol

### Request

OPTIONS \* HTTP/1.1  
Host: www.inc.com

### Reply

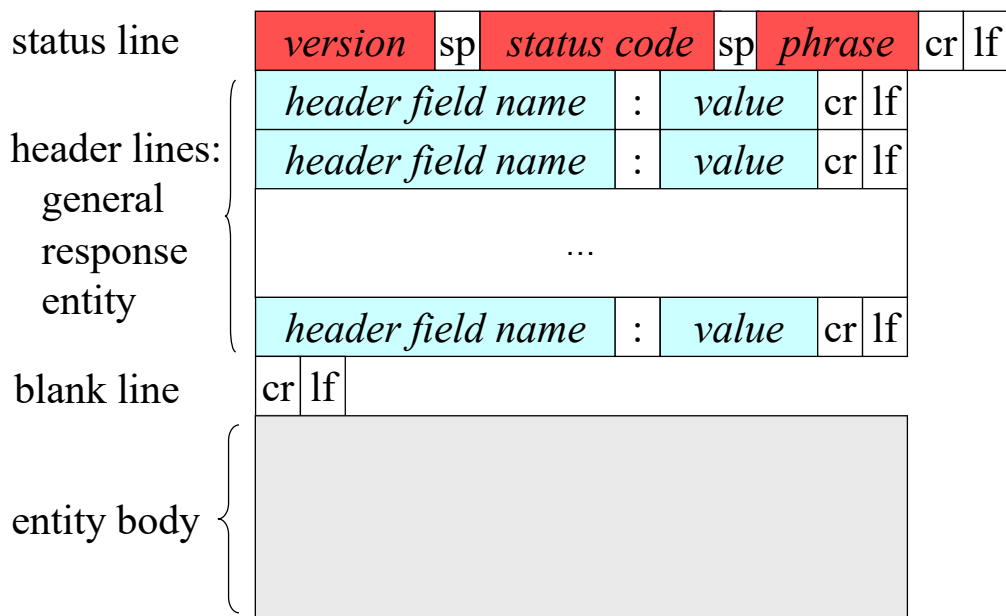
200 Ok  
Allow: OPTIONS, GET, HEAD, POST, PUT  
Accept-Ranges: bytes  
Accept-Encoding: gzip

- In the case of a specific URI, informs which methods the server can provide for the object
  - Applied without actually retrieving the full resource

# Method: HEAD

- HEAD: metadata for a resource
- Just like a GET operation, except that the server does not return the actual object requested
  - Saves network bandwidth and server resources
  - E.g., verify that object exists (not necessarily to get it)
- Gives proxies a way to see if an object has changed without actually retrieving the full object
  - E.g., examine Last-Modified or Content-Length

## General Response Message



# Response (E.g.)

```
HTTP/1.1 200 OK
Date: Sat, 3 Mar 2007 00:31:35 GMT
Server: Apache/1.3.27 tomcat/1.0
Last-Modified: Fri, 8 Nov 2002 23:40:01 GMT
Set-Cookie: A=joe; path=/; expires=Wed, 09-Nov-2009 23:12:40 GMT
ETag: "20e-6c4b-3da21b51"
Accept-Ranges: bytes
Content-Length: 369
Keep-Alive: timeout=5, max=300
Connection: Keep-Alive
Content-Type: text/html

<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title>ABC Center</title>
<meta http-equiv="Content-Type" content="text/html;
  charset=iso-8859-1">
<meta http-equiv="refresh" content="1800; URL=http://abc.com/">
<meta name="DESCRIPTION" content="The ABC Centre">
</head>
<body bgcolor="#FFFFFF">
<a name= "body.html"></a>
</body>
</html>
```


## HTTP Headers

General Headers	
Date	Cache Control
Pragma	Connection
	Trailer
	Transfer-Encoding
	Upgrade
	Via
	Warning

- Apply only to a message itself
- Can appear both in requests and responses

Request Headers	
Authorization	Accept
From	Accept-Charset
If-Modified-Since	Accept-Encoding
Referer	Accept
User-Agent	Language
	Expect
	Host
	If-Match
	If-None-Match
	If-Range
	If-Unmodified-Since
	Max-Forwards
	Proxy-Authorization
	Range
	TE

Send additional info or specify constraints

 HTTP/1.0 & HTTP/1.1

 HTTP/1.1



# HTTP Headers (Cont'd)

Response Headers	
Location	Accept-Ranges
Server	Age
WWW-Authenticate	ETag
	Proxy-Authenticate
	Retry-After
	Vary

Send info about response/server

Entity Headers	
Allow	Content-Language
Content-Encoding	Content-Location
Content-Length	Content-MD5
Content-Type	Content-Range
Expires	
Last-Modified	

Info about entity or request resource

	HTTP/1.0 & HTTP/1.1
	HTTP/1.1

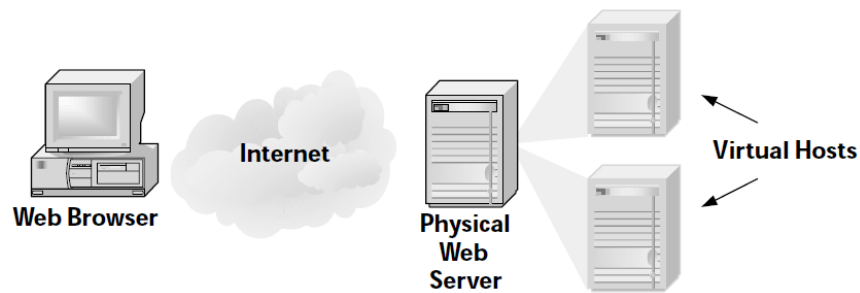
## HTTP/1.1

- Enhancements:
  - New methods: PUT, DELETE, OPTIONS, TRACE
  - Persistent connections
  - Pipelining requests
  - Compression encoding
  - Range requests
  - Extends cacheability headers
  - The Upgrade header
- First defined in RFC 2068
- Later replaced by RFC 2616, and finally revised in RFCs 7230 to 7235



# Host Header

- For backward compatibility cannot change the format of the Request-Line to mandate the server hostname
  - Host: www.host.com:80
- Only required header by 1.1; “400” error returned otherwise
- Supports virtual hosting: *multiple domains with a single IP address*



## Message Transmission

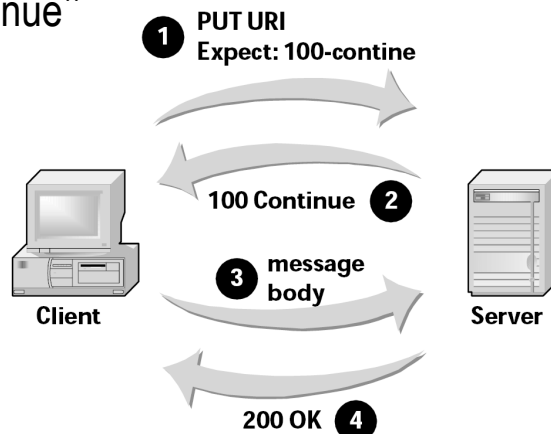
- Content-length field (1.0) vs. chunked encoding (ends with zero-length chunk)
  - Content-Length required by 1.1 to support persistent connection
  - Content-Length for dynamically produced content?
- Range Requests
  - Range, If-Range, Content-Range
- Expect/Continue
  - E.g., prepare for a large put
- Other headers
  - Via: hop-by-hop behavior

# Reliable Correct Caching

- Semantically Transparent Caching
  - Affects the behavior of neither client nor origin server
- Caching only improves performance
- Client receives exactly the same response from cache/origin server
- Consider other cases: If-Unmodified-Since
  - Useful with PUTs
- Age (seconds) since validated with or generated by origin server
  - Estimated by sender (i.e., caches)
- Extend date/time stamp with opaque entity tag (ETag)

## Expect Header

- Client tells a server that it expects a certain behavior
- It's defined as an extensible header
- But the only currently defined use for it is if a client expects a server to send a 100 Continue status
  - E.g., “Expect: 100-continue”



## Continue Response

- Informational status codes: 1xx
- Some HTTP requests (e.g., PUT or POST methods) carry request bodies, which may be arbitrarily long
  - If server is not willing to accept the request, perhaps because of an authentication failure, it would be a waste of bandwidth to transmit such a large request body
- HTTP/1.1 uses status code, 100 (Continue), to inform the client that the request body should be transmitted
- When this mechanism is used, the client first sends its request headers, then waits for a response
  - E.g., Receive 100 (Continue) or 401 (Unauthorized)

## Chunked Content

- Chunked Transfer-Encoding
  - If a server wants to start sending a response before knowing its total size, it may use chunked transfer encoding
  - Breaks the complete response into smaller chunks and sends them in series
- Contains the following header:
  - Transfer-Encoding: chunked
- All HTTP/1.1 clients must be able to receive chunked messages

## Chunked Content (cont' d)

- A chunked message body contains:
  - A series of chunks
  - Followed by a line with "0" (zero)
  - Followed by optional footers (similar to headers)
  - A blank line
- Each chunk consists of two parts:
  - A line with the size of the chunk data, in hex, possibly followed by a semicolon and extra parameters you can ignore (none are currently standard)
  - Data

## Chunk Trailers

- Chunking solves another problem related to sender-side message buffering
- Some header fields, such as Content-MD5 (i.e., checksum over the message body), cannot be computed until after the message body is generated
- In HTTP/1.0, the use of such header fields required the sender to buffer the entire message
- In HTTP/1.1, a chunked message may include a trailer after the final chunk
- A trailer is simply a set of one or more header fields

# Chunked Content (E.g.)

- E.g.:

```
HTTP/1.1 200 OK\r
Date: Sat, 01 Jan 2000 23:59:50 GMT\r
Content-Type: text/plain\r
Transfer-Encoding: chunked\r
Trailer: Expires\r
\r
1a\r
ABCDEFGHIJKLMNOPQRSTUVWXYZ\r
2\r
12\r
0\r

Expires: Tue, 20 May 2008 23:59:50 GMT
```
- Length value of chunk in hex
- Content-Length not used
- Trailer header can be used

## Partial Content

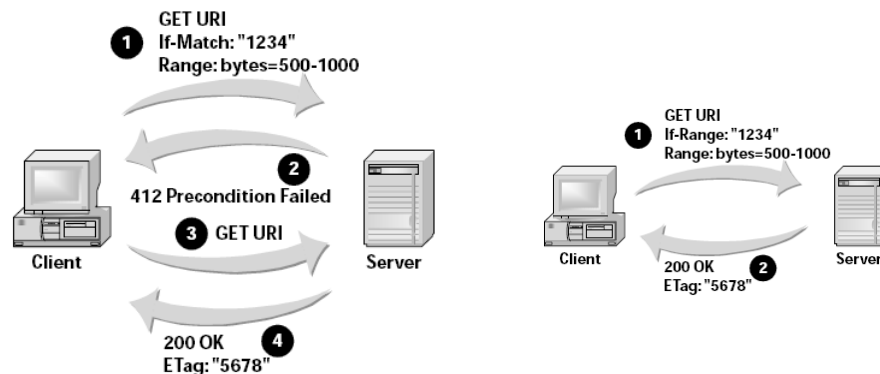
- A type of conditional request to get some bytes of the resource
- A client may need only part of a resource, e.g.,
  - Display just the beginning of a long document
  - Continue downloading a file after a transfer was terminated in mid-stream
  - May want to use or cache only some portion of an object or entity
- HTTP/1.1 range requests allow a client to request portions of a resource
  - Supports only ranges of bytes
  - Client makes a range request by including the Range header in its request, specifying one or more contiguous ranges of bytes

## (cont' d)

- Range requests:
  - **Range:** allows client to request a range
  - **Content-Range:** specifies where a partial entity should be inserted (e.g., Content-Range: 0-65535/83028576 )
  - **If-Range:** if the entity is unchanged, send the part(s) that are missing; otherwise, send the entire new entity
- May also use a date to ask to return the partial range if the resource has not been modified since the specified date
- 206 partial content response code

## If-Match vs. If-Range Header

- If-Match performs the operation if the same entity
- If-Range can send the the same or entire new entity if changed
  - Partial object returned if existing part is still valid, otherwise the full object is returned
- If-\* headers considered only if the request would otherwise return a 200 OK status



# Server Cache Directives

- Reduce redundant data transfers with client caching
  - Avoid round-trip latency and extra network load
- Use Cache-Control header (HTTP/1.1)
- Rich set of control directives, for example:
  - Do not save this content anywhere: no-store
  - Do not serve this content again without revalidating: no-cache (but it can actually be cached!)
- Hold on to this content for a bounded time
  - Cache-control: max-age=14400
- Don't hold on to it past a certain time
  - Expires: Fri, 1 May 2020 10:00:00 GMT

# Resource Validation

- Set timestamps for your content
  - Use **Last-Modified**
  - Use **Date:** header (required)
- Servers can also associate IDs with web objects
  - Objects may vary based on time-of-day or geographic location (e.g., yahoo.com)
  - Represents object ID (e.g., for version control) on the server
    - Strong: resources are identical byte for byte
    - Weak: resources are equivalent “semantically”
  - IDs are called Entity Tags, in **ETag:** header,  
ETag: “cc678-12d12-66394036”  
ETag: W/“abcd” (weak ETag)

## Resource Validation (cont'd)

- ETag acts as opaque id for the cache or client, typically used with **If-Match** and **If-None-Match**: request header
- Can also help with editing resources with PUTs
  - Prevent conflicts
  - Update a resource if ETag matches the condition
- With timestamps or ids, caches or clients can perform conditional GETs
  - If-Modified-Since: Fri, 1 May 2020 10:00:00 GMT
  - If-None-Match: “cc678-12d12-66394036”

\*\*\*



# Connection Management

- TCP works best for long-lived connections
- Original HTTP design used a new TCP connection for each request
  - Each request incurred the cost of setting up a new TCP connection
  - At least one round-trip time across the network, plus several overhead packets
- Since most Web interactions are short (the median response message size is about 4 Kbytes)
  - TCP connections seldom get past the “slow-start” region
  - Fail to maximize their use of the available bandwidth

## (cont'd)

- Web pages frequently have embedded resources, e.g., images, sometimes many of them, and each one is retrieved via a separate HTTP request
  - The average number of embedded resources has risen over the years to 100s
- The use of a new TCP connection for each resource retrieval serializes the display of the entire page on the connection-setup latencies for all of the requests
  - Browsers use parallel TCP connections to compensate for this serialization, but the possibility of increased congestion limits the utility of this approach

# Transport Issues

- One TCP connection for request/response: simple
  - Non-persistent connection (HTTP 0.9, 1.0)
- Multiple parallel connection to improve response
  - Typically, 5 or 6 connections
  - Multiple 3-way handshake, slow starts
  - Several extra round trips delays added to transfer
  - Can end up grabbing more of the bottleneck bandwidth
  - Different objects in a page arrive independently
- Response time may still be poor
  - TCP loss recovery dominated by timeouts in small windows

## Transport Issues (cont' d)

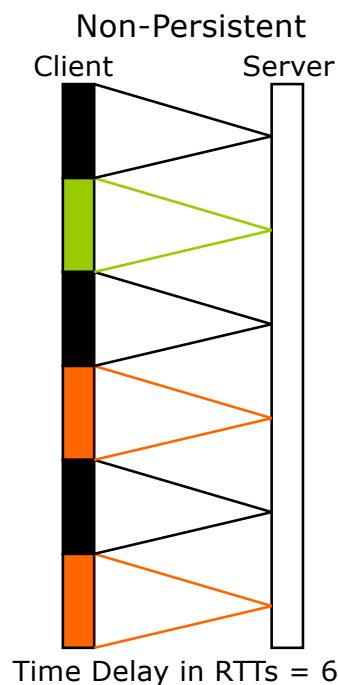
- Short transfers are poor in TCP
  - Still in slow start phase
  - Loss recovery is poor with small window (dominated by timeouts)
- Many more TCP connections are used
  - OS must process and allocate host resources for each TCP connection – increases overhead
  - Servers often incur the TIME\_WAIT connection state
- To resolve these problems, the use of persistent connections and the pipelining of requests was added

# Persistent Connection

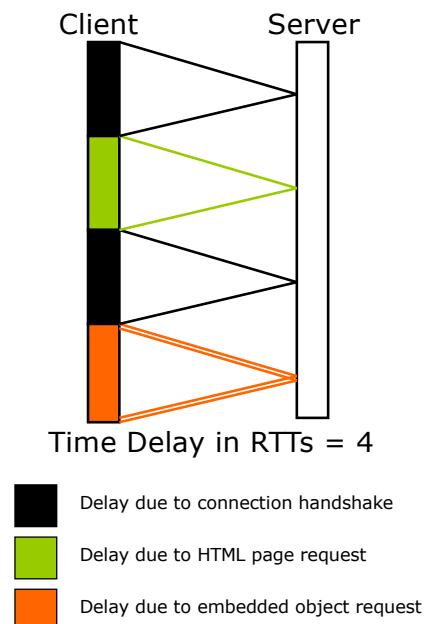
- Multiplex multiple transfers onto one TCP connection
  - Serialize transfers → client makes next request only after previous response received
- How to demultiplex requests/responses
  - Content-length and delimiter
  - Wait for entire response and then use content-length
- Pipeline requests
  - Send multiple requests before responses return
- Headers:
  - Connection: Keep-Alive
  - Keep-Alive: max=10, timeout=60
- Default behavior in HTTP/1.1

## Non-Persistent Connection (E.g.)

A web page with 2 embedded objects



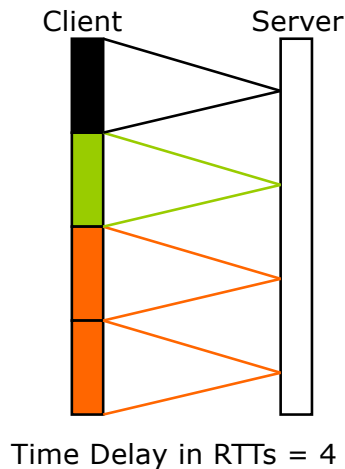
Non-Persistent with Pipelining



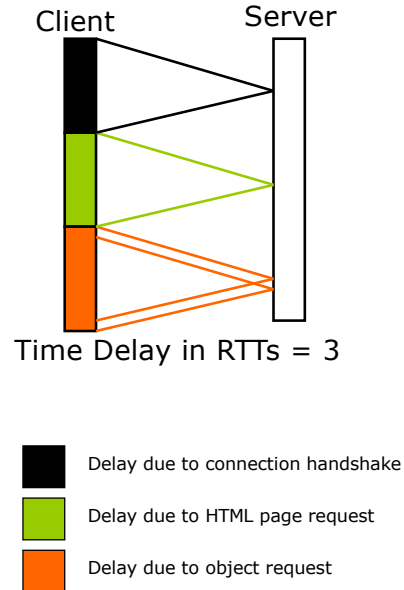
# Persistent Connection (E.g.)

A web page with 2 embedded objects

Persistent w/o pipelining



Persistent with pipelining



## Modern Web Page

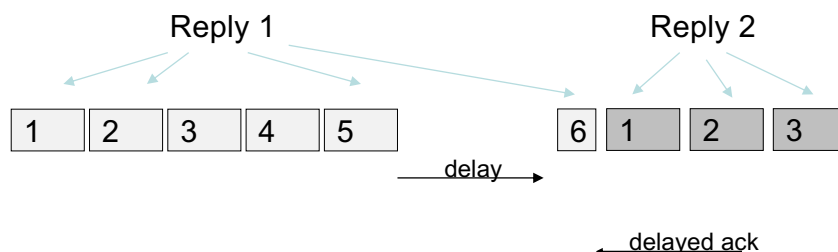
- An average Web page is composed of more than 90 resources (e.g., images, javascript)
- Resource fetched often from more than 15 distinct hosts
- Most of the data flows are smaller than 15KB
- Bursty transfers
- TCP was designed for long-lived connections
- Challenges to reduce the latency

# HTTP over TCP

- TCP algorithms may interfere negatively with HTTP transfers
- Nagle's algorithm: limit the number of small packets sent by TCP from HTTP server
- Delayed acks: receiver may delay transmission to utilize piggybacking on an outgoing data packet
  - May increase the latency in HTTP messages
  - Traffic pattern is generally asymmetrical between server and client

# HTTP over TCP

- Negative performance impact with *persistent* connections
- May delay the transfer the last packet of HTTP message
  - E.g., payload from server smaller than MSS
- Can disable the algorithms by setting socket option TCP\_NODELAY with persistent connections
  - But care should be given to avoid small TCP segments



# HTTP Connection and TCP

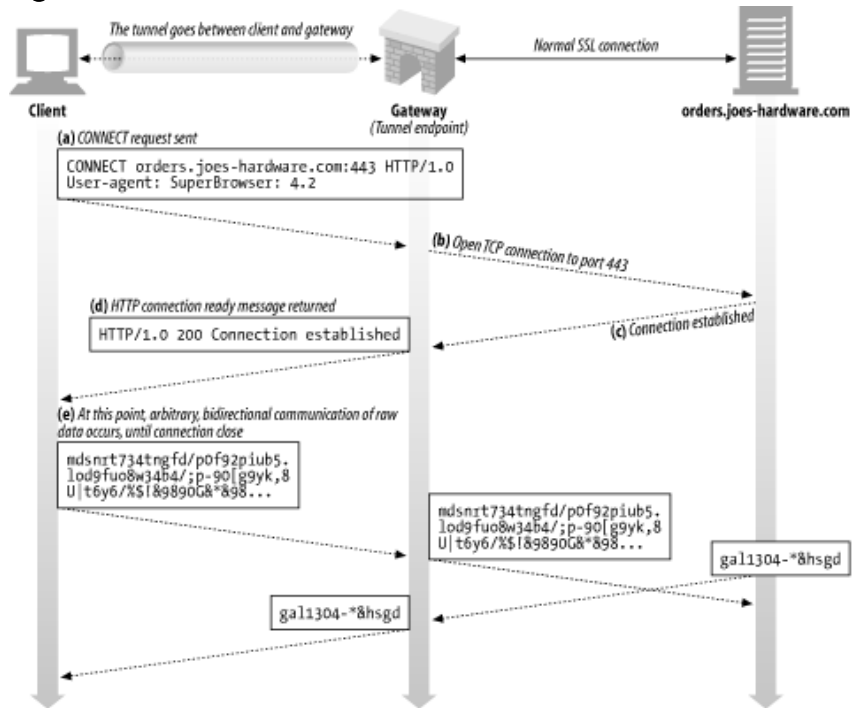
- HTTP connection types: persistent and/or pipelined
- Limitations:
  - Cancelling of an on-going request by user
    - I.e., cancel all pipelines request not processed yet?
    - Completing it for a large object wastes network resources
  - Wait for new TCP connection before making next request
  - No way for the HTTP server to acknowledge cancellation

## Web Tunnels

- Allow sending non-HTTP traffic through HTTP connections
- Common use:
  - Embed non-HTTP traffic inside an HTTP connection so it can be sent through firewalls that allow only web traffic
  - Any TCP-based protocol
- Established using HTTP's CONNECT method
  - Create a TCP connection to an arbitrary destination server
  - Response has no content-type header

# CONNECT and SSL (E.g)

- Using CONNECT to establish an SSL tunnel



## CONNECT Proxy (E.g.)

request

```
( CONNECT www.examples.com:443 HTTP/1.1\r\n
  User-Agent: Wget/1.19.4 \r\n
  Host: www.examples.com:443\r\n
  \r\n
```

response

```
( HTTP/1.1 200 Connection established\r\n
  \r\n
```

# HTTP/1.1 Feature Highlights

HTTP 1.1 Feature	Implication
Persistent Connection	Lowers number of connection setups
Pipelining	Shortens inter-arrival of requests
Expires	Lowers number of validations
Entity Tags	Lowers frequency of validations
Max-Age, Max-Stale etc	Changes frequency of validations
Range Request	Lowers bytes transferred
Chunked Encoding	Lowers user perceived latency
Expect/Continue	Lowers error response/bandwidth
Host Header	Reduces proliferation of IP addresses

## Upgrading

- In order to ease the deployment of incompatible future protocols, HTTP/1.1 includes the new Upgrade request header
- By sending the Upgrade header, a client can inform a server of the set of protocols it supports as an alternate means of communication
- The server may choose to switch protocols, but this is not mandatory
- Can be used to switch to HTTP/2 clear-text (“h2c”)