A Comprehensive Analysis on HTTP and Related Topics

Submitted to Dr. Muhammad Nomani Kabir

CSE 3711 : Computer Networks Theory (A) Summer-24

A Comprehensive Analysis on HTTP and Related Topics

Submitted to Dr. Muhammad Nomani Kabir
CSE 3711 : Computer Networks Theory (A) Summer-24

Mohammad Sameer Ahmed [redacted] **Mymuna Nourin** [redacted] **Zobaer Ibn Razzaque** [redacted] **Robiul Awoul Robin** [redacted] **Md. Sadman Haque** [redacted]

Abstract - This report is created as part of the assignment for the replacement of the midterm exam of Summer-24. This assignment covers the following topics 1. HTTP and TCP Connection: A Fundamental Overview 2. Variations Among HTTP Methods 3. Web Browser Cookies: A Technical Overview 4. Web Cache : All You Need to Know 5. Mathematical Demonstration of Real Life Application in Non Persistent & Persistent HTTP. The report is in serial with this. This report dives into various depths in all the mentioned topics, to give the reader a better understanding of them.

1. HTTP and TCP Connection: A Fundamental Overview

HTTP: Application Layer Protocol

As an application layer protocol, HTTP acts as the primary mechanism for clients (usually web browsers) to request data from servers.A typical interaction involves the client sending an HTTP request and the server responding with the requested data.

Client-Server Model

HTTP operates on a client-server model.In this model, a client, such as a browser like Firefox or Safari, sends a request to a server to retrieve a resource (e.g., a web page).The server running a web server like Apache responds to the client by sending the requested web objects.

This back-and-forth exchange forms the basis of web browsing.HTTP has evolved since its inception, with major steps in its standardization including HTTP/1.0 (RFC 1945) and HTTP/1.1 (RFC 2068).The transition from HTTP/1.0 to HTTP/1.1 marks significant improvements, particularly in the way the protocol handles connections over TCP.

1.1 The Role of TCP in HTTP

1.1.1 Establishing a TCP Connection

HTTP, as an application layer protocol, requires a reliable transport mechanism to ensure that data is delivered to its destination correctly and in order.This reliability is provided by TCP (Transmission Control Protocol), a transport layer protocol.When a client initiates an HTTP session, it must first establish a TCP connection with the server.

This process involves several steps:

1.The client initiates the TCP connection: The client initiates the connection by sending a SYN (synchronize) packet to the server.This packet represents a request to establish a connection.

2.Server acknowledges with SYN-ACK: The server responds with a SYN-ACK packet, acknowledging the client's request to establish a connection.

3.Client sends ACK: The client sends an ACK (acknowledgement) packet back to the server, completing the three-step negotiation process.

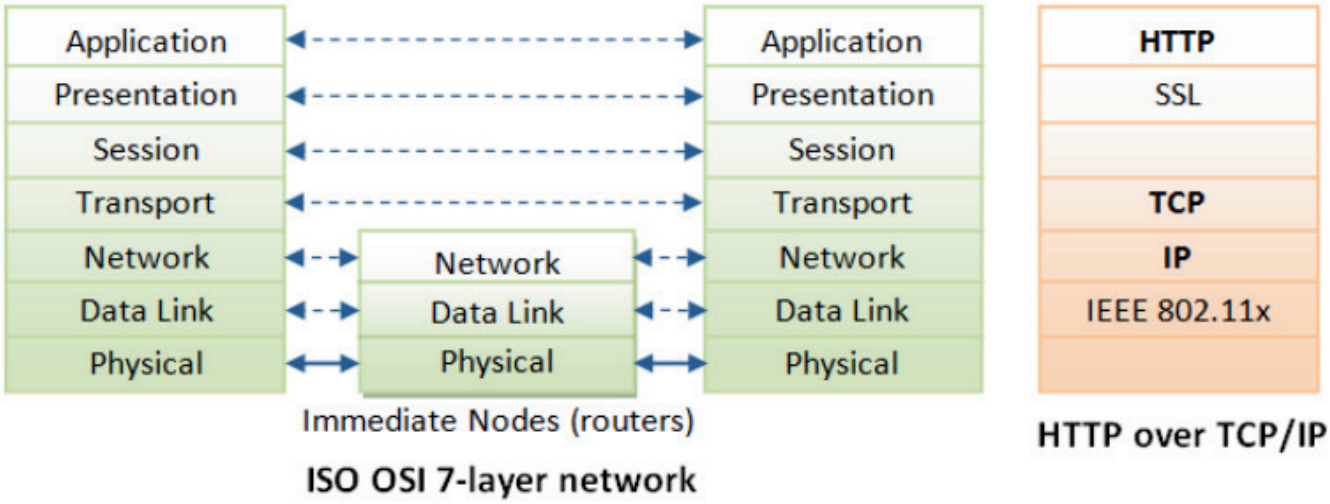
At this point, the TCP connection is established and data can begin to be transmitted.Once the connection is established, the client and server exchange HTTP messages, which are essentially application layer protocol messages transmitted over the TCP connection.

1.1.2 Port 80 and Communication

The standard TCP port used by HTTP is port 80.Once a connection is established, HTTP requests and responses pass through this port, ensuring smooth communication between the client and the server.

Request: The client sends an HTTP request message (e.g.a GET request to retrieve a web page).

Response: The server processes this request and returns an HTTP response (e.g.an HTML page or JSON data).



1.1.3 Closing the TCP Connection

When an HTTP session ends, the TCP connection is terminated through a process called the four-way handshake.The client or server can initiate this process by sending a FIN packet, which signals that it wants to close the connection.The other party responds with an ACK packet and sends its own FIN packet to confirm the end of the connection.
Finally, the party that initiated the connection responds with an ACK, which completes the shutdown.

1.2 HTTP as a Stateless Protocol

HTTP is defined as a stateless protocol, meaning that each HTTP request from the client to the server is independent of the previous request.The server does not retain any information about past requests.This design decision simplifies the protocol but has important implications for stateful interactions, such as user sessions.

1.2.1 Complexity of Maintaining State

Stateful protocols are inherently more complex.They need to keep track of user interactions, which requires storing history or past state.In the event of a client- or server-side failure, the protocol must ensure that state is restored and adjusted, which can introduce additional overhead.In HTTP, the stateless nature simplifies communication but requires additional techniques to keep track of state across multiple requests.
For example, cookies and session management mechanisms are often used to allow the server to “remember” information between multiple requests from the same client.

2. Variations Among HTTP Methods

2.1 Uploading Form Input

Forms are a fundamental component of web pages, allowing users to submit data to a server. When a user submits form data, it is uploaded and processed by the server using one of two primary methods: POST or GET

2.1.1 POST Method

The POST method is commonly used when submitting form data to a web server. In this approach:

- Form Input: The input data from the user is included in the”body” of the HTTP request.
- Usage: This method is preferred for transmitting large amounts of data, including sensitive information like passwords or personal details.
- Security: Since data is not visible in the URL, POST offers a level of security, especially when using encryption (HTTPS).
- Limitations: One disadvantage of POST is that it cannot be bookmarked or cached, making it unsuitable for requests that need to be revisited frequently.

2.1.2 GET Method

The URL method uses the HTTP GET request, which uploads the form input directly in the URL field of the request. This method has its own set of features:

- Form Input: Input data is appended as query parameters to the URL itself (e.g., `www.example.com/page?name=value`).

- Usage: GET is best suited for retrieving data without making permanent changes on the server. It is ideal for short, simple inputs, such as search queries.
- Visibility: Data is visible in the URL, making it easier to bookmark and share, but also less secure, as it can be viewed in browser history.
- Limitations: GET has character limits for URLs, restricting the amount of data that can be transmitted. Additionally, sensitive data should not be sent via GET due to its visibility.

2.1.3 Comparing the Two Methods

- Security: POST is more secure than GET, as data is hidden within the request body. GET transmits data in the URL, which can be viewed and logged.
- Size: POST has no practical size limit for form data, while GET is limited by URL length restrictions.
- Caching and Bookmarking: GET allows caching and bookmarking, making it useful for read-only operations. POST does not support this, making it better for operations like submitting data to a database.

2.2 Methods Types

In the world of web communications, the Hypertext Transfer Protocol (HTTP) plays a critical role in how clients and servers interact. Over time, new versions of HTTP have introduced additional methods to enhance functionality. Below is a breakdown of some key HTTP methods and how they work.

2.2.1 HTTP/1.0 Methods

1. GET : GET method is used to request data from a specified resource on the server. It sends parameters through the URL.

- Example: When visiting a web page, the browser sends a GET request to retrieve the page's content.

2. POST : Unlike GET, which passes parameters in the URL, POST sends data in the body of the request.

- Example: When filling out a registration form, the submitted data is sent to the server using POST.

3. HEAD : HEAD method functions similarly to GET but with a key difference, it only retrieves the headers, not the entire content. This method is useful for checking metadata about a resource.

- Example: To check if a page has been updated without downloading it, you can use a HEAD request.

2.2.2 HTTP/1.1 Methods

1. GET, POST, HEAD: These methods remain unchanged from HTTP/1.0 and are foundational to web communication.

2. PUT: The PUT method is used to upload a file or data to the server. It places the file in the entity body of the request and uploads it to the specified path in the URL. PUT is generally used for updating an existing resource or creating a new resource if it doesn't already exist.

- Example: When uploading an image or updating a file on the server, PUT is the method commonly used.

3. DELETE: DELETE method is used to remove a resource on the server. The file or resource specified in the URL is deleted upon successful execution of this method.

- Example: A request to delete a specific file from the server would use the DELETE method.

2.3 Differences Between HTTP/1.0 and HTTP/1.1

The key difference between HTTP/1.0 and HTTP/1.1 lies in the additional functionality provided by HTTP/1.1. While HTTP/1.0 only supports GET, POST, and HEAD, HTTP/1.1 introduces methods like PUT and DELETE, enabling more robust interaction with resources on the server.

3. Web Browser Cookies: A Technical Overview

Introduction to Cookies

Web browser cookies are small text files that are created by a web server and stored on a user's device by the web browser. These cookies are used to store stateful information about the user's interactions with a website, enabling the server to remember the user across different sessions or within the same session. Cookies are essential for maintaining continuity in web applications, such as keeping users logged in, remembering their preferences, or tracking their activity for analytics and advertising purposes.

3.1 The Purpose and Functionality of Cookies 01

Cookies facilitate a stateful experience in the otherwise stateless HTTP protocol. They allow servers to store user-specific information and retrieve it on subsequent requests, thereby personalizing the web experience. Key purposes include:

- Session Management: Cookies track the user's session on the website, such as keeping them logged in across multiple pages.
- Personalization: Cookies store user preferences, like language settings or theme choices.
- Tracking and Analytics: Cookies are used to monitor user behavior, gather analytics, and deliver targeted advertisements.

3.2 Types of Cookies

Sl No.	Type	Definition	Usage	Example
1	Session Cookies	Session cookies are temporary cookies that are erased once the user closes their web browser	Commonly used for maintaining user sessions, such as keeping a user logged in during their browsing session	An e-commerce website might use a session cookie to remember the items added to a shopping cart as the user navigates through the site
2	Persistent Cookies	Persistent cookies remain on the user's device until they expire or are manually deleted by the user. These cookies have a specific expiration date	Used for storing login information, language preferences, or tracking long-term user behavior	A website might use a persistent cookie to keep a user logged in for 30 days unless they log out manuall
3	First-Party Cookies	First-party cookies are set by the domain that the user is visiting directly	These cookies are used by the website to store user preferences, manage login states, and remember other session-related information	If you visit example.com, a first-party cookie might be used to remember your language preference
4	Third-Party Cookies	Third-party cookies are set by a domain other than the one the user is visiting, often through ads embedded content, or social media plugins	Primarily used for tracking users across different websites, enabling targeted advertising and cross-site analytics	Visiting example.com might set a third-party cookie from adnetwork.com that tracks your browsing behavior for personalized ads

3.3 How Cookies Work

When a user visits a website, the server sends a cookie to the user's browser. The browser stores this cookie, and when the user revisits the same website, the browser sends the cookie back to the server. This process allows the server to recognize the user and tailor the user experience based on the stored data.

3.3.1 Example of a Cookie Interaction:

1. Client Request: The user visits a website, and the browser sends a request to the server.
2. Server Response: The server responds with the requested content and includes a cookie in the response headers.
3. Cookie Storage: The browser stores the cookie according to the rules specified in the response, such as the expiration date, path, and domain.
4. Subsequent Requests: On subsequent visits to the website, the browser automatically includes the cookie in its request headers, allowing the server to retrieve the stored data and customize the response.

3.3.2 Security and Privacy Concerns :

While cookies enhance user experience, they also pose potential security and privacy risks. Since cookies can track browsing behavior, they can be exploited for malicious purposes if not properly secured. Key concerns include:

- Cross-Site Scripting (XSS): Attackers can inject malicious scripts into web pages that access and misuse cookies.
- Cross-Site Request Forgery (CSRF): Exploits the trust a site has in the user's browser, tricking it into sending unauthorized requests by leveraging stored cookies.
- Tracking and Profiling: Third-party cookies can track user activities across different websites, leading to extensive profiling and targeted advertising.

3.3.3 Managing Cookies

Modern browsers offer various settings for managing cookies, allowing users to:

- View and Delete Cookies: Users can view stored cookies and delete them if desired, providing control over their personal data.
- Block Third-Party Cookies: Browsers can block cookies from domains other than the one displayed in the address bar, reducing tracking across websites.
- Clear Browsing Data: Users can clear all stored cookies and browsing data to maintain privacy.

4. Web Cache : All You Need to Know

Web caching is a technique used to store copies of web resources temporarily, allowing faster retrieval and reduced load times for users accessing websites. By storing data closer to the end-user or within the web server, caching improves both performance and resource efficiency.

4.1 How Web Caching Works

When a user accesses a web page, their browser sends a request to the server. If the requested resource (such as HTML, images, or JavaScript files) is already stored in the cache, it can be retrieved locally without needing to contact the server again. This reduces latency and conserves bandwidth.

There are two key types of web caches:

1. Browser Cache: This cache is stored on the user’s device. When a page is visited, the browser saves certain resources, like images and stylesheets, so that subsequent visits are quicker.
2. Proxy Cache: A proxy cache stores web resources on intermediary servers located between the client and the web server. Proxy caches are commonly used by ISPs and organizations to reduce bandwidth usage.

4.2 Types of Web Cache

- Client-Side Cache (Browser Cache)
  - Description: Resources are cached in the user’s browser, reducing the number of requests made to the server for frequently accessed content.
  - Usage: Browser cache is best used for static resources like images, CSS files, or JavaScript files, which don’t change often.
  - Expiration: This cache is controlled by HTTP headers like Cache-Control and Expires, which tell the browser how long to keep a resource cached.

- Server-Side Cache (Reverse Proxy Cache)
  - Description: A reverse proxy server caches resources on behalf of a web server, responding to requests without having to query the server.
  - Usage: Server-side cache is used to reduce server load and improve response times for frequently accessed resources.
  - Examples: Popular tools like Varnish Cache or NGINX serve as reverse proxy caches, sitting between the client and the web server.

4.3 Cache-Control HTTP Headers

HTTP caching is managed through specific headers in the request and response. These headers control how and when the browser or other caching mechanisms can store a resource.

1. Cache-Control: Defines the caching policies such as public, private, no-store, or max-age.
   - Example: Cache-Control: max-age=3600 tells the browser to cache the resource for 1 hour.
2. Expires: Specifies an exact date and time when the cached resource should expire. After this, the browser must fetch a fresh copy from the server.
3. ETag: A unique identifier assigned to a specific version of a resource. If the content changes, the ETag changes, and the cache is refreshed.

Example:
ETag: "34a64df551429fcc55e4d42a148795d9f25f89d4"

4.4 Benefits of Web Caching

1. Faster Load Times: Since cached resources are served from a nearby cache instead of making a full round-trip to the server, users experience faster page loads.
2. Reduced Bandwidth Usage: Web caching minimizes the need for repeated requests to the server, saving on bandwidth.
3. Lower Server Load: Serving content from a cache reduces the number of requests that need to be processed by the server, allowing it to handle more users at once.
4. Improved Scalability: Web caches help websites scale better by distributing the load across multiple caches rather than putting all the pressure on the origin server.

4.5 Downsides of Web Caching

- Stale Content: Cached content may become outdated if not refreshed properly, leading to users seeing old data.
- Cache Invalidation: Determining when to clear the cache or refresh it with new content can be tricky. If done too aggressively, it might negate the benefits of caching.
- Complex Configuration: Setting up and managing caching systems, particularly server-side caching, requires proper configuration and ongoing management to avoid performance bottlenecks.

4.6 Cache Invalidation

Cache invalidation is the process of updating or removing cached resources when they become outdated. This can be managed using:

1. Time-Based Expiry: Setting specific expiration times for cached resources using HTTP headers.
2. Conditional Requests: The browser checks with the server using If-Modified-Since or If-None-Match headers to see if a cached resource has changed. If not, the cache is used; otherwise, the fresh content is fetched.

4.7 Content Delivery Networks (CDNs)

CDNs are large-scale distributed caching systems that store copies of web content on servers around the world. When a user requests a resource, the CDN serves it from the nearest location, improving speed and reducing latency. CDNs make use of caching extensively to improve global web performance.

Examples of popular CDNs include:

- Cloudflare
- Akamai
- Amazon CloudFront

5. Mathematical Demonstration of Real Life Application in Non Persistent & Persistent HTTP

Persistent HTTP:
Persistent HTTP, is a communication feature where a single TCP connection is reused to send and receive multiple HTTP requests and responses, reducing the overhead of establishing new connections.

Non-persistent HTTP:
Non-persistent HTTP is a communication method where a new TCP connection is established for each HTTP request and closed after the response is received, leading to increased connection overhead for multiple requests.

RTT (Round Trip Time):
time for a small packet to travel from client to server and back.

Propagation Delay:
Propagation delay is the time it takes for a signal to travel from the sender to the receiver across a communication medium.

Transmission Delay:
Transmission delay is the time required to push all the bits of a data packet onto the communication link, determined by the packet size and the transmission rate of the link.

Mathematical Terms:
Assume that you have a base HTML file with 30 embedded images, images & base file are small enough to fit in one TCP segment. How many RTT are required to retrieve base file & images under the following conditions:
(Assume RTT dominates all other time)

1. Non-Persistent connection without parallel connection.
2. Non-Persistent connection with 10 parallel connections.
3. Persistent connection without pipelining.
4. Persistent connection with pipelining.

Explanation:
2 RTT is the initial required connection, one for TCP connection and one for HTML base file.
Total time = 2 RTT + transmit time

5.1 Types and Worked out Examples of Round Trip Time Calculation

5.1.1 Non-Persistent connection with no parallel connection:
For each image, 2 RTT are required—one for the TCP connection and one for the image to send.
So,
transmit time for 30 images = 2 * (30 RTT) = 60 RTT
Total time = 2 RTT + 60 RTT = 62 RTT

5.1.2 Non-persistent connection with 10 parallel connections:
Here, 10 images can be sent simultaneously.
So, for 30 images, it requires -> 2 * (30 / 10) = 6 RTT
Total time = 2 RTT + 6 RTT = 8 RTT

5.1.3 Persistent connection without pipelining:
Here, the TCP connection is required again and again.
So, for 30 images, it requires -> 30 RTTs
Total time = 2 RTT + 30 RTT = 32 RTT

5.1.4 Persistent connection with pipelining:
Since it is a persistent connection, the TCP connection is not required again and again. Pipelining means in one packet only images that can fit can be sent.
In a pipelining connection, we can send all images in 1 RTT.
Total time = 2 RTT + 1 RTT = 3 RTT.

5.2 Real Life Application with Mathematical Terms

Suppose a website contains 1 HTML file, 10 CSS files, and 5 JavaScript files.

File Type	Size
HTML	100kB
CSS	30kB
JavaScript	50kB

The RTT required for connection setup is 0.3 sec. The bandwidth between the server and the client is 0.5 Gbps. The distance between the client and the server is 5 km. Propagation speed is 200 m/microsec.

Now, calculate the response time required to retrieve the base files and images under the following conditions:
1. Non-persistent HTTP without parallel connection.
2. Persistent HTTP without pipelining.

Solution:

1. Non-persistent HTTP without parallel connection:
RTT = 0.3 sec
Transmission Rate = 0.5*10^9 bps
Propagation Delay = (5*10^3) / (2*10^8) = 2.5*10^-5 sec

For , HTML = ((2*0.3) + ((100*10^3) / (0.5*10^9))) * 1
= 0.6002 sec

For , CSS = ((2*0.3) + ((30*10^3) / (0.5*10^9))) * 10
= 6.0006 sec

For , JS = ((2*0.3) + ((50*10^3) / (0.5*10^9))) * 5
= 3.0005 sec

So, Non-persistent HTTP without parallel connection response time
= ((2*0.3) + (2.5*10^-5) + 0.6002 + 6.0006 + 3.0005)
= 10.201 sec

2. Persistent HTTP without pipelining:
RTT = 0.3 sec
Transmission Rate = 0.5*10^9 bps
Propagation Delay = 2.5*10^-5 sec

For , HTML = (0.3 + ((100*10^3) / (0.5*10^9))) * 1
= 0.3002 sec

For , CSS = (0.3 + ((30*10^3) / (0.5*10^9))) * 10
= 3.0006 sec

For JS = (0.3 + ((50*10^3) / (0.5*10^9))) * 5 = 1.5005

So, Persistent HTTP without pipelining response time
= (0.3 + (2.5*10^-5) + 0.3002 + 3.0006 + 1.5005)
= 5.101 sec

From the above topic, it covers the differences between persistent and non-persistent HTTP connections, focusing on how they handle multiple file transfers in a webpage. It explains key networking terms like RTT, propagation delay, and transmission delay. Through mathematical examples, it shows how non-persistent HTTP incurs more delays by opening separate connections for each file, while persistent HTTP improves efficiency by reusing connections. It then applies these concepts to calculate the response times for a website containing HTML, CSS, and JavaScript files, illustrating the performance differences between the two connection types.