

HTTP Overview Explanation



Applicatin Layer.pdf
PDF

explain slide 17,18 and 19 in great details and clearly
explain everything

Sure! I'll explain **slides 17, 18, and 19** from the **Application Layer** chapter in great detail, clearly covering all key points.

Slide 17: Web and HTTP — Overview

Main points:

- A **web page** is composed of multiple objects.
- Each object can be stored on **different Web servers**.
- An object can be:
 - An HTML file (the main text of a webpage),
 - A JPEG image,
 - A Java applet,
 - An audio file, etc.
- A web page consists of a **base HTML file** which contains references (URLs) to several other objects.
- Each object on the web page is addressable via a **URL (Uniform Resource Locator)**.

Example URL:

```
bash
```

```
www.someschool.edu/someDept/pic.gif
```

- This URL contains two main parts:
 - **Host name:** `www.someschool.edu`
 - **Path name:** `/someDept/pic.gif`

Detailed Explanation:

When you open a webpage on your browser, the browser does not just load one single file. Instead, it loads a base HTML file (which has the structure and content of the page) plus many other files like images, videos, scripts, etc. These files may be located on different servers.

Each item (object) on the webpage is uniquely identified by a URL. For example, if the webpage contains images, those images have their own URLs and might be hosted on completely different servers than the HTML page itself.

The base HTML file acts like a blueprint, telling the browser where to find and download all the other files that make the webpage complete.

Slide 18: HTTP Overview

Main points:

- **HTTP (Hypertext Transfer Protocol)** is the application-layer protocol used by the Web.
 - It follows a **client-server model**:
 - The **client** is typically the browser that requests, receives, and displays Web objects.
 - The **server** is the Web server that sends objects in response to client requests.
 - Communication is done by sending **HTTP requests** from client to server and receiving **HTTP responses** back.
 - Examples of clients: Browsers like Safari (on iPhone), Firefox (on PC).
 - Examples of servers: Web servers running software like Apache.
-

Detailed Explanation:

HTTP is the language that browsers and web servers use to talk to each other.

- When you type a website address or click a link, your browser (client) sends an HTTP request message to the web server asking for a webpage or other content.

- The web server receives the request, finds the requested object (like an HTML file, image, or video), and sends it back in an HTTP response message.
- Your browser then displays the content you requested.

Because HTTP is designed around this request-response model, it's simple and efficient for delivering web content.

Slide 19: HTTP Overview (continued)

Main points:

- HTTP uses **TCP** (Transmission Control Protocol) as the underlying transport protocol.
- The client initiates a **TCP connection** to the server on **port 80** (default port for HTTP).
- The server accepts the TCP connection.
- HTTP messages (requests and responses) are exchanged between the client and server over this TCP connection.
- After the exchange, the TCP connection is closed.
- HTTP is a **stateless protocol**:
 - The server does **not keep any information** about past client requests.
 - This makes HTTP simpler, but it also means that every request is treated independently.
- Maintaining state (like login sessions) requires more complex protocols or additional mechanisms because HTTP itself does not track past interactions.

Detailed Explanation:

TCP provides reliable, ordered, and error-checked delivery of a data stream. When you visit a webpage:

1. Your browser establishes a TCP connection with the server (like opening a telephone call).
2. Your browser sends an HTTP request over this connection.

3. The server processes the request and sends back an HTTP response.
4. The connection is closed.

Because HTTP is stateless, it treats every request as new and independent. For example, if you load a webpage and then click a link on that page, the server doesn't remember you or your previous request unless extra tools like cookies or sessions are used.

Statelessness makes HTTP simpler and scalable but requires developers to implement additional features if the server needs to remember clients over multiple interactions (like shopping carts or login status).

Summary:

- **Slide 17** explains the composition of web pages as multiple objects each referenced by URLs.
- **Slide 18** introduces HTTP as the protocol enabling browsers (clients) and servers to communicate and exchange those web objects.
- **Slide 19** details how HTTP relies on TCP connections, explains the request-response process, and highlights HTTP's stateless nature.

If you want, I can also explain the next slides or clarify any of these points more deeply!

explain slide 20,21,22 and 23 in great details and clearly explain everything

Absolutely! Here's a detailed and clear explanation of **slides 20, 21, 22, and 23** from the Application Layer chapter.

Slide 20: HTTP Connections: Two Types

This slide explains the two main types of HTTP connections:

1. Non-persistent HTTP

- For each object (file/image/script) requested, a **new TCP connection** is opened.
- The process:
 - Open TCP connection.
 - Send HTTP request.
 - Server sends back **at most one object**.
 - TCP connection is then closed.
- If a webpage has multiple objects (images, CSS, JS files, etc.), the client must open **multiple TCP connections** — one per object.
- This can cause inefficiency because opening and closing TCP connections repeatedly adds latency and overhead.

2. Persistent HTTP

- A single TCP connection is opened between the client and server.
 - Multiple HTTP requests and responses can be sent over the **same** TCP connection.
 - The TCP connection remains open, reducing the overhead of repeatedly opening and closing connections.
 - This results in **better performance** especially for webpages with many referenced objects.
 - After all required objects are transferred, the TCP connection is closed.
-

Detailed Explanation:

- **Non-persistent HTTP** wastes time opening and closing TCP connections repeatedly, which slows down loading a webpage.
 - **Persistent HTTP** reduces latency by keeping the connection open and sending multiple requests/responses back and forth.
 - Most modern browsers and web servers use persistent HTTP (HTTP 1.1 and beyond) by default.
-

Slide 21: Non-persistent HTTP: Example

This slide shows a step-by-step example of how non-persistent HTTP works:

1. The **HTTP client** initiates a TCP connection to the server on port 80.
2. The client sends an HTTP request message, asking for a specific object (e.g., `/someDepartment/home.index`).
3. The server accepts the TCP connection and receives the request.
4. The server sends the requested object back to the client through the TCP connection.
5. The server closes the TCP connection.
6. The client receives the object and processes it (e.g., displays the HTML).
7. If the received object (HTML) contains references to other objects (e.g., 10 JPEG images), the client repeats steps 1-6 for each object, opening a new TCP connection every time.

Detailed Explanation:

- For each referenced object on the webpage, the client performs the full TCP connection cycle.
- This creates a delay because of the overhead in establishing and tearing down connections.
- This approach was used in early web designs but is inefficient for pages with many resources.

Slide 22: Non-persistent HTTP: Example (continued)

This slide continues the example and highlights:

- Steps 1–5 are repeated for **each of the 10 JPEG objects** referenced by the base HTML file.
- This means for one webpage with multiple objects, there will be multiple TCP connection setups and teardowns, significantly increasing total load time.

Detailed Explanation:

- If a webpage references 10 images, the client will open 10 separate TCP connections sequentially.
 - Each TCP connection setup involves a handshake (which takes time), plus the data transfer.
 - Closing and reopening TCP connections repeatedly adds unnecessary latency.
-

Slide 23: Non-persistent HTTP Response Time

This slide quantifies the time taken for a non-persistent HTTP connection to deliver one object.

Key terms:

- **RTT (Round-Trip Time):** Time taken for a packet to travel from client to server and back.
 - **File transmission time:** Time to actually send the requested object data.
-

Response Time Calculation for one object:

1. **One RTT to initiate TCP connection:** TCP setup handshake requires one round-trip between client and server.
 2. **One RTT for HTTP request and response:** Client sends request, server sends back the initial part of the response.
 3. **File transmission time:** The time to transfer the entire file.
-

Total response time for each object = 2 RTT + file transmission time

Detailed Explanation:

- The first RTT is TCP's three-way handshake to establish the connection.

- The second RTT is the time for the HTTP request message to reach the server and the first bytes of the response to reach back.
 - The final time is for the actual data to transfer, which depends on the file size and connection speed.
 - For pages with many objects, this delay is multiplied by the number of objects, making non-persistent HTTP inefficient.
-

Summary of Slides 20-23:

- **Slide 20:** Introduced two HTTP connection types—non-persistent (one TCP connection per object) and persistent (one TCP connection for multiple objects).
 - **Slides 21 & 22:** Walked through how non-persistent HTTP works in practice, with each object requiring a full TCP connection cycle.
 - **Slide 23:** Detailed the response time for non-persistent HTTP, showing that each object download requires 2 RTTs plus the time to transmit the file, which can add significant delay for multiple objects.
-

If you want, I can continue explaining how **persistent HTTP** works in subsequent slides or clarify anything here!

slide 23 and 24

Certainly! Here's a detailed explanation of **slides 23 and 24** from the Application Layer chapter:

Slide 23: Non-persistent HTTP Response Time

This slide breaks down the **time it takes to get a response** for one object using non-persistent HTTP.

Key concepts:

- **RTT (Round-Trip Time):**
The time it takes for a small packet to travel from the client to the server and back again.
- **Steps for one object download in non-persistent HTTP:**
 1. **One RTT to initiate TCP connection:**
TCP requires a handshake to establish a connection, which takes one full round trip between client and server.
 2. **One RTT for HTTP request and initial response:**
The client sends the HTTP request, and the server sends the first few bytes of the HTTP response back.
 3. **File transmission time:**
The actual time it takes to transfer the full object (file) once the response has started.

Formula for response time per object:

$$\text{Response time} = 2 \times \text{RTT} + \text{File transmission time}$$

Detailed Explanation:

- For each object (like an image or HTML file), non-persistent HTTP requires a fresh TCP connection.
 - Because of the TCP handshake and request/response exchange, there are 2 RTT delays before the actual file starts transmitting.
 - This adds latency, especially noticeable when a webpage contains many objects, each requiring separate TCP connections.
-

Slide 24: Persistent HTTP (HTTP 1.1)

This slide explains how **persistent HTTP** improves over non-persistent HTTP.

Problems with non-persistent HTTP:

- Requires **2 RTTs per object**, causing a lot of delay.
- Operating System overhead for creating and closing TCP connections repeatedly.

- Browsers often open **multiple parallel TCP connections** to speed things up, but this adds complexity.

Persistent HTTP (introduced in HTTP/1.1):

- The TCP connection is **kept open** after the initial HTTP response.
 - Multiple HTTP requests and responses can be sent over this **single open TCP connection**.
 - The client can send new HTTP requests as soon as it encounters referenced objects in the page.
 - Because of this:
 - Only **one RTT may be needed for all referenced objects**, significantly reducing response time.
 - Cuts the total download time roughly in half compared to non-persistent HTTP.
 - The TCP connection is eventually closed once all objects are transferred.
-

Detailed Explanation:

- Instead of opening and closing TCP connections repeatedly, persistent HTTP keeps the connection alive.
 - This reduces the overhead from TCP handshakes and connection teardowns.
 - It allows the client to pipeline multiple HTTP requests quickly and efficiently.
 - Persistent connections are now the default in modern browsers and servers due to these efficiency gains.
-

Summary of Slide 23 and 24:

- **Slide 23** quantifies the delay in non-persistent HTTP, showing that each object needs 2 RTTs plus transmission time, making it slow for pages with many objects.
- **Slide 24** introduces persistent HTTP, which keeps TCP connections open, allowing multiple objects to be downloaded with fewer RTTs and less overhead, greatly improving web performance.

Let me know if you want me to explain the next slides or if you want a simple summary!