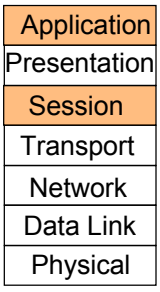
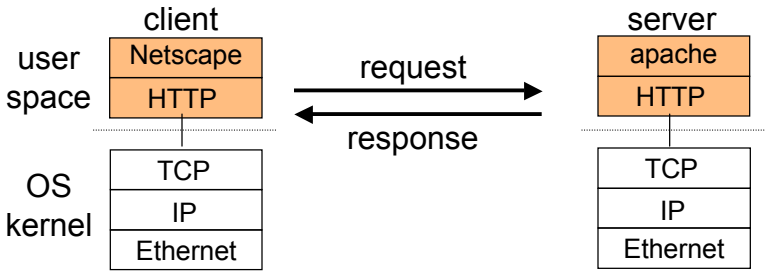
# CSE 461
# HTTP and the Web

## This Lecture

- HTTP and the Web (but not HTML)
- Focus
  - How do Web transfers work?
- Topics
  - HTTP, HTTP1.1
  - Performance Improvements
    - Protocol Latency
    - Caching



# Web Protocol Stacks




- To view the URL http://server/page.html the client makes a TCP connection to port 80 of the server, by it's IP address, sends the HTTP request, receives the HTML for page.html as the response, repeats the process for inline images, and displays it.

# HTTP Request/Response

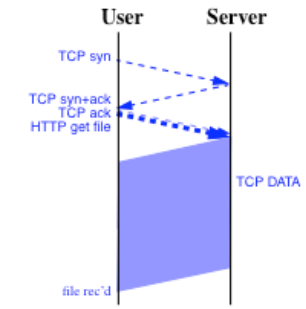
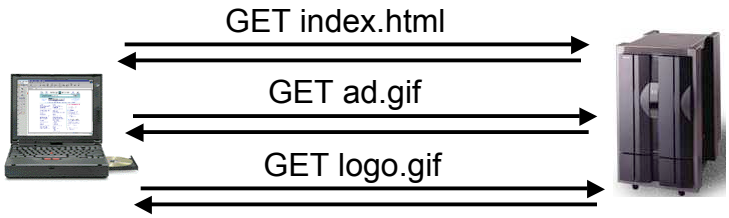



FIGURE 3. HTTP File Transfer

| | |
|---|---|
| 1 RTT | channel OPEN |
| 0.5 RTT | send request |
| 0.5 RTT | file starts to arrive |
| Ftrans | time to transmit the file |
| --- | |
| 2 RTT + Ftrans | |
| = time to get a file in HTTP | |

# Simple HTTP 1.0




- HTTP is a tiny, text-based language
- The GET method requests an object
- There are HTTP headers, like "Content-Length:", etc.
- Try "telnet server 80" then "GET index.html HTTP/1.0"
  - Other methods: POST, HEAD,… google for details

# HTTP Request/Response in Action

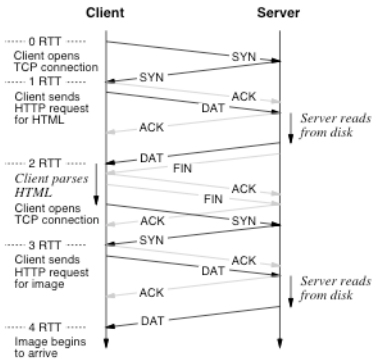



**Figure 3-1:** Packet exchanges and round-trip times for HTTP

Problem is that:

- Web pages are made up of many files
  - Most are very small (< 10k)
- files are mapped to connections

For each file

- Setup/Teardown
  - Time-Wait table bloat
- 2RTT "first byte" latency
- Slow Start+ AIMD Congestion Avoidance

The goals of HTTP and TCP protocols are not aligned.

- Implications

# TCP Behavior for Short Connections Over Slow Networks

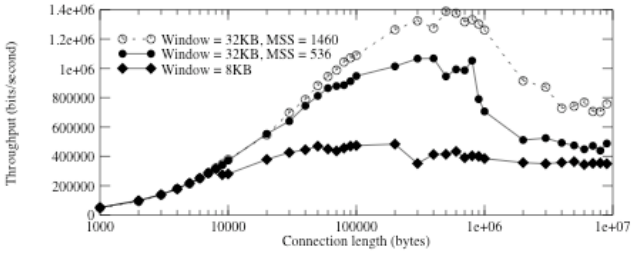



**Figure 3-2:** Throughput vs. connection length, RTT = 70 msec

Figure 3-2 shows that, in the remote case, using a TCP connection to transfer only 2 Kbytes results in a throughput less than 10% of best-case value. Even a 20 Kbyte transfer achieves only about 50% of the throughput available with a reasonable window size. This reduced throughput translates into increased latency for document retrieval. The figure also shows that, for this 70 msec RTT, use of too small a window size limits the throughput no matter how many bytes are transferred.

RTT=70ms

# It's the RTT

RTT=1ms

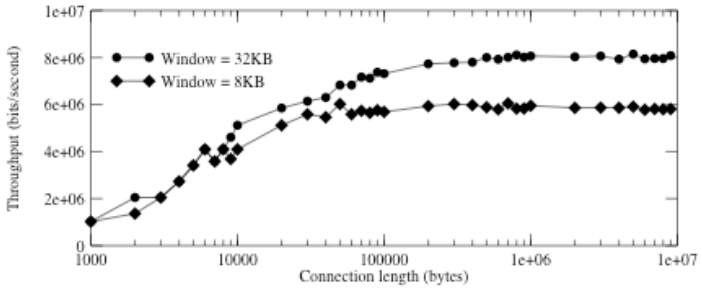
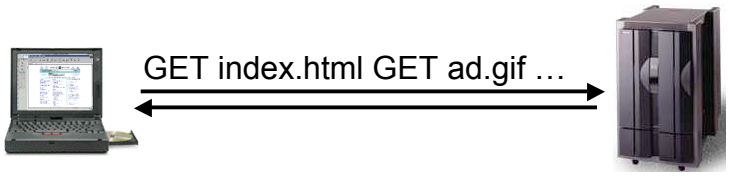



**Figure 3-3:** Throughput vs. connection length, RTT near 0 msec

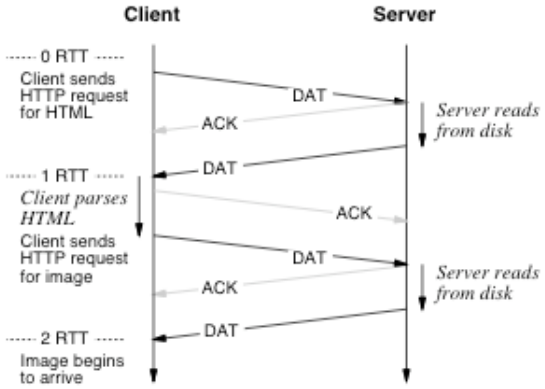
*No slow start here (ULTRIX LAN)*

# HTTP1.1: Persistent Connections




- Bright Idea: Use one TCP connection for multiple page downloads (or just HTTP methods)
- Q: What are the advantages?
- Q: What are the disadvantages?
  - *Application layer multiplexing*

# HTTP/1.1



# Effect of Persistent HTTP

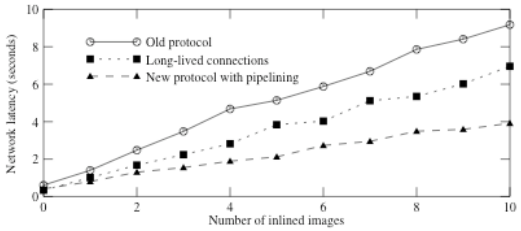



**Figure 6-1:** Latencies for a remote server, image size = 2544 bytes

Image size=2544

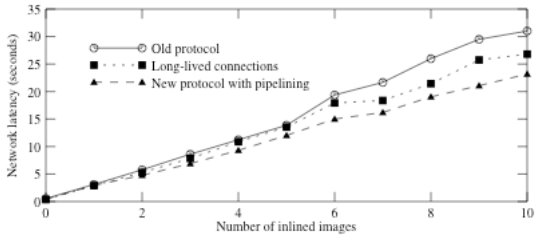
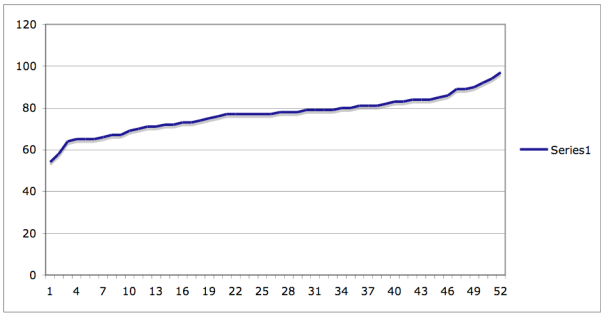



**Figure 6-2:** Latencies for a remote server, image size = 45566 bytes

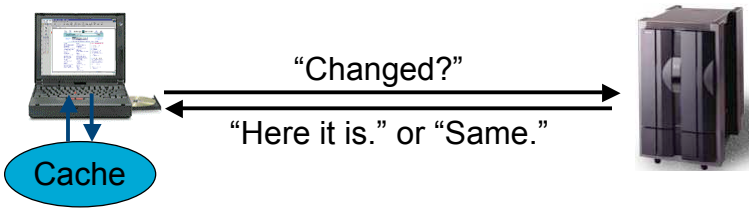
Image size=45566



# Caching

- *It is faster and cheaper to get data that is closer to here than closer to there.*
- *"There" is the origin server. 2-5 RTT*
- *"Here" can be:*
  - *Local browser cache (file system) (1-10ms)*
  - *Client-side proxy (institutional proxy) (10-50)*
  - *Content-distribution network (CDN -- "cloud" proxies) (50-100)*
  - *Server-side proxy (reverse proxy @ origin server) (2-5RTT)*

# Browser Caches

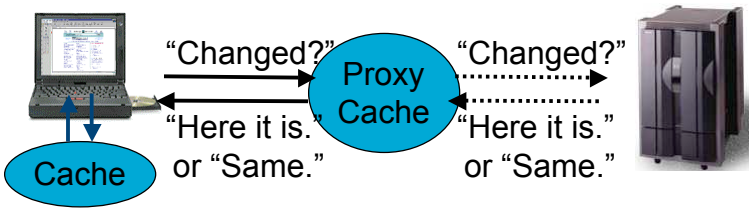



- Bigger win: avoid repeated transfers of the same page
- Check local browser cache to see if we have the page
- GET with If-Modified-Since makes sure it's up-to-date
- Q: What are the advantages and disadvantages?

# Consistency and Caching Directives

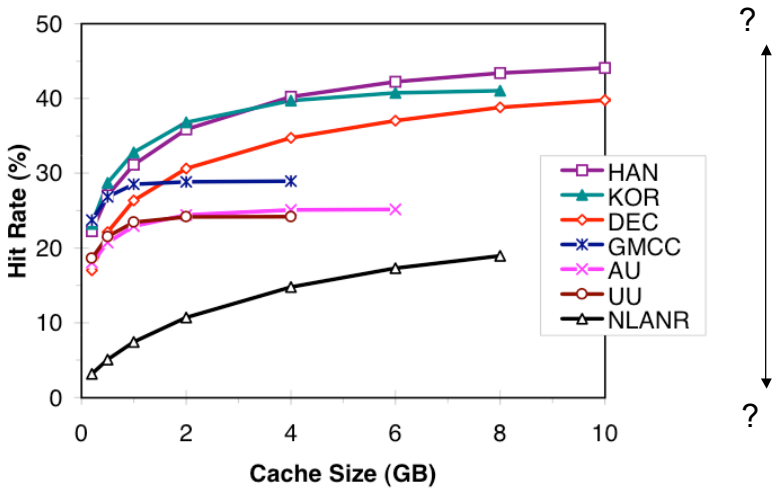
- Key issue is knowing when cached data is fresh/stale
  - Otherwise many connections or the risk of staleness
- Browsers typically use heuristics
  - To reduce server connections and hence realize benefits
  - Check freshness once a "session" with GET If-Modified-Since and then assume it's fresh the rest of the time
  - Possible to have inconsistent data.
- Caching directives provide hints
  - Expires: header is basically a time-to-live
  - Also indicate whether page is cacheable or not

# Proxy Caches




- Insert further levels of caching for greater gain
- Share proxy caches between many users (not shown)
  - If I haven't downloaded it recently, maybe you have
- Your browser has built-in support for this

# Proxy Cache Effectiveness




# Hit Rate Follows Request Rate

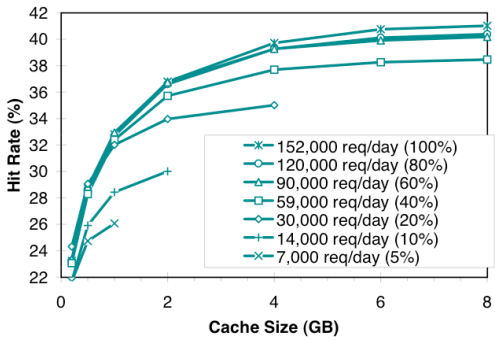



Figure 3: Cache hit rate for KOR as a function of cache size for a range of request rates.

# Sharing, Not Locality, Drives Effectiveness

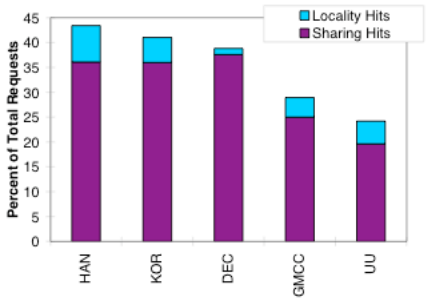



Figure 9: Hit rate divided into hits due to sharing and due to locality of a single client.

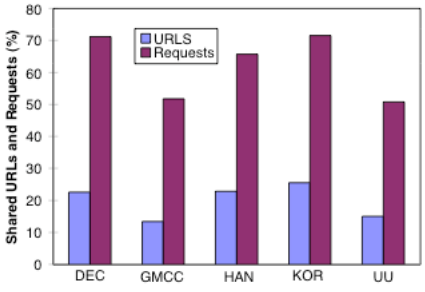
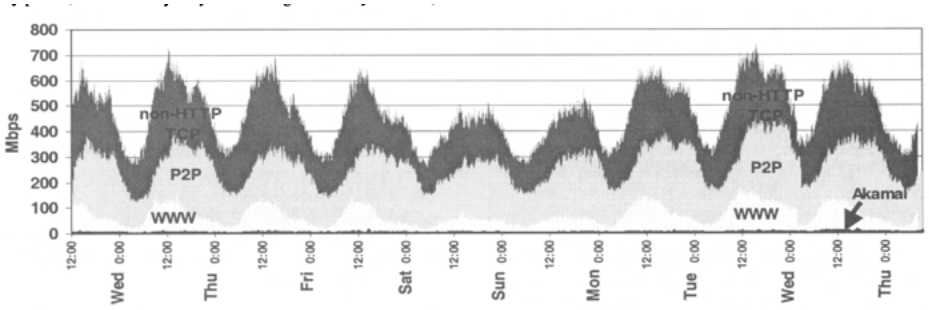



Figure 10: The percent of a total URLs in a trace requested by two or more clients and the percent of total requests to these shared objects.

# The Trends

- HTTP Objects are getting bigger
- But Less important



# Key Concepts

- HTTP and the Web is just a shim on top of TCP
  - Sufficient and enabled rapid adoption
  - Many "scalability" and performance issues now important