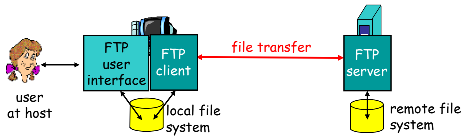
CS640: Introduction to Computer Networks

Aditya Akella

Lecture 4 -
Application Protocols, Performance

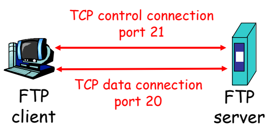
Applications
FTP: The File Transfer Protocol



- Transfer file to/from remote host
- Client/server model
  - *Client:* side that initiates transfer (either to/from remote)
  - *Server:* remote host
- ftp: RFC 959
- ftp server: port 21

FTP: Separate Control, Data Connections

- Ftp client contacts ftp server at port 21, specifying TCP as transport protocol
- Two parallel TCP connections opened:
  - Control: exchange commands, responses between client, server.
    "out of band control"
  - Data: file data to/from server



- Server opens data connection to client
  - Exactly one TCP connection per file requested.
  - Closed at end of file
  - New file requested → open a new data connection
- Ftp server maintains "state": current directory, earlier authentication

HTTP Basics

- HTTP layered over bidirectional byte stream
  - Almost always TCP
- Interaction
  - Client sends request to server, followed by response from server to client
  - Requests/responses are encoded in text
- Contrast with FTP
  - Stateless
    - Server maintains no information about past client requests
      - There are some caveats
  - In-band control
    - No separate TCP connections for data and control

Typical HTTP Workload (Web Pages)

- Multiple (typically small) objects per page
  - Each object a separate HTTP session/TCP connection
- File sizes
  - Why different than request sizes?
  - Heavy-tailed (both request and file sizes)
    - "Pareto" distribution for tail
    - "Lognormal" for body of distribution

Non-Persistent HTTP

http://www.cs.wisc.edu/index.html

1. Client initiates TCP connection
2. Client sends HTTP request for index.html
3. Server receives request, retrieves object, sends out HTTP response
4. Server closes TCP connection
5. Client parses index.html, finds references to 10 JPEGs
6. Repeat steps 1—4 for each JPEG
   (can do these in parallel)

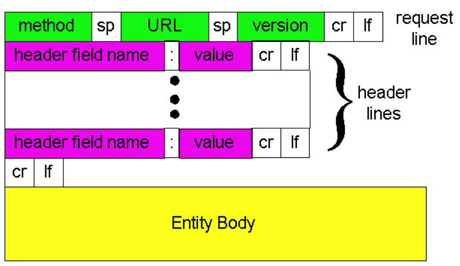
Issues with Non-Persistent HTTP

- Two "round-trip times" per object
  - RTT will be defined soon
- Server and client must maintain state per connection
  - Bad for server
  - Brand new TCP connection per object
- TCP has issues starting up ("slow start")
  - Each object face to face these performance issues
- HTTP/1.0

The Persistent HTTP Solution

- Server leaves TCP connection open after first response
  - W/O pipelining: client issues request only after previous request served
    - Still incur 1 RTT delay
  - W/ pipelining: client sends multiple requests back to back
    - Issue requests as soon as a reference seen
    - Server sends responses back to back
      - One RTT for all objects!
- HTTP/1.1

HTTP Request



HTTP Request

- Request line
  - Method
    - GET – return URI
    - HEAD – return headers only of GET response
    - POST – send data to the server (forms, etc.)
  - URL
    - E.g. /index.html if no proxy
    - E.g. http://www.cs.cmu.edu/~akella/index.html with a proxy
  - HTTP version

HTTP Request

- Request header fields
  - Authorization – authentication info
  - Acceptable document types/encodings
  - From – user email
  - If-Modified-Since
  - Referrer – what caused this page to be requested
  - User-Agent – client software
- Blank-line
- Body

HTTP Request Example

GET /~akella/index.html HTTP/1.1
Host: www.cs.wisc.edu
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Connection: Keep-Alive

HTTP Response

- Status-line
  - HTTP version
  - 3 digit response code
    - 1XX – informational
    - 2XX – success
      - 200 OK
    - 3XX – redirection
      - 301 Moved Permanently
      - 303 Moved Temporarily
      - 304 Not Modified
    - 4XX – client error
      - 404 Not Found
    - 5XX – server error
      - 505 HTTP Version Not Supported
  - Reason phrase

HTTP Response

- Headers
  - Location – for redirection
  - Server – server software
  - WWW-Authenticate – request for authentication
  - Allow – list of methods supported (get, head, etc)
  - Content-Encoding – E.g x-gzip
  - Content-Length
  - Content-Type
  - Expires
  - Last-Modified
- Blank-line
- Body

HTTP Response Example

HTTP/1.1 200 OK
Date: Thu, 14 Sep 2006 03:49:38 GMT
Server: Apache/1.3.33 (Unix) mod_perl/1.29 PHP/4.3.10 mod_ssl/2.8.22 OpenSSL/0.9.7e-fips
Last-Modified: Tue, 12 Sep 2006 20:43:04 GMT
ETag: "62901bbe-161b-45071bd8"
Accept-Ranges: bytes
Content-Length: 5659
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<data data data>

Cookies: Keeping "state"

Many major Web sites use cookies
→ keep track of users
→ Also for convenience: personalization, passwords etc.

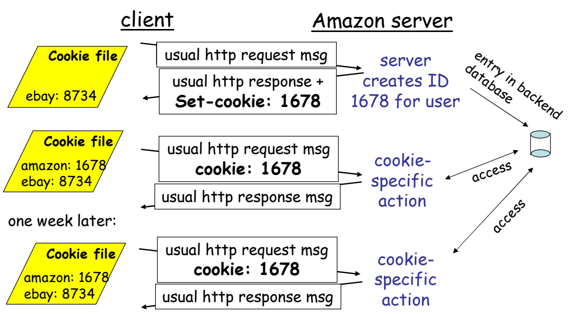
Four components:

1) Cookie header line in the HTTP response message
2) Cookie header line in HTTP request message
3) Cookie file kept on user's host and managed by user's browser
4) Back-end database at Web site

Example:

- Susan accesses Internet always from same PC
- She visits a specific e-commerce site for the first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

Cookies: Keeping "State" (Cont.)



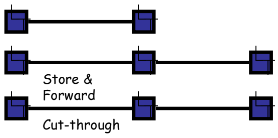
Performance Measures

- Latency or delay
  - How long does it take a bit to traverse the network
- Bandwidth
  - How many bits can be crammed over the network in one second?
- Delay-bandwidth product as a measure of capacity

Packet Delay: One Way and Round Trip

- Sum of a number of different delay components.
- Propagation delay on each link.
  - Proportional to the length of the link
- Transmission delay on each link.
  - Proportional to the packet size and 1/link speed
- Processing delay on each router.
  - Depends on the speed of the router
- Queuing delay on each router.
  - Depends on the traffic load and queue size
- This is one-way delay
  - Round trip time (RTT) = sum of these delays on forward and reverse path

Ignoring processing and queuing…



Prop + xmit

2*(Prop + xmit)

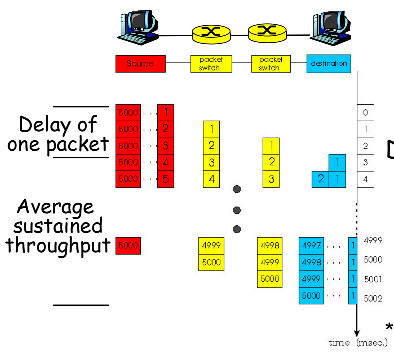
2*prop + xmit

Aside: When does cut-through matter?

Routers have finite speed (processing delay)

Routers may buffer packets (queueing delay)

Ignoring processing and queuing…



$\text{Delay}^* + \dfrac{\text{Size}}{\text{Throughput}}$

Units: seconds + bits/(bits/seconds)

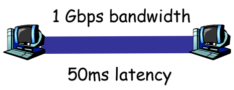
* For first bit to arrive

Some Examples

- How long does it take to send a 100 Kbit file? 10Kbit file?

Latency \ Throughput	100 Kbit/s	1 Mbit/s	100 Mbit/s
500 μsec	0.1005	0.0105	0.0006
10 msec	0.11	0.02	0.0101
100 msec	0.2	0.11	0.1001

Bandwidth-Delay Product



- Product of bandwidth and delay (duh!)
  - What is it above?
- What does this indicate?
  - #bytes sender can xmit before first byte reaches receiver
  - Amount of "in flight data"
- Another view point
  - B-D product == "capacity" of network from the sending applications point of view
  - Bw-delay amount of data "in flight" at all time → network "fully" utilized

TCP's view of BW-delay product

- TCP expects receiver to acknowledge receipt of packets
- Sender can keep up to RTT * BW bytes outstanding
  - Assuming full duplex link
  - When no losses:
    - 0.5RTT * BW bytes "in flight", unacknowledged
    - 05RTT * BW bytes acknowledges, acks "in flight"

Extra slides

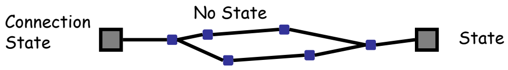
Internet Architecture

- Background
  - "The Design Philosophy of the DARPA Internet Protocols" (David Clark, 1988).
- Fundamental goal: "Effective techniques for multiplexed utilization of existing interconnected networks"
- "Effective" → sub-goals; in order of *priority*:
  1. ***Continue despite loss of networks or gateways***
  2. Support multiple types of communication service
  3. Accommodate a variety of networks
  4. Permit distributed management of Internet resources
  5. Cost effective
  6. Host attachment should be easy
  7. Resource accountability

Survivability

- If network disrupted and reconfigured
  - Communicating entities should not care!
  - This means:
    - Transport interface only knows "working" and "not working"
    - Not working == complete partition.
    - Mask all transient failures
- How to achieve such reliability?
  - State info for on-going conversation must be protected
  - Where can communication state be stored?
    - If lower layers lose it → app gets affected
    - Store at lower layers and replicate
      - But effective replication is hard

Fate Sharing



- Lose state information for an entity if (and only if?) the entity itself is lost
  - Protects from intermediate failures
  - Easier to engineer than replication
  - Switches are stateless
- Examples:
  - OK to lose TCP state if one endpoint crashes
    - NOT okay to lose if an intermediate router reboots