Persistent HTTP: Padmanabhan and Mogul [Padmanabhan95a]

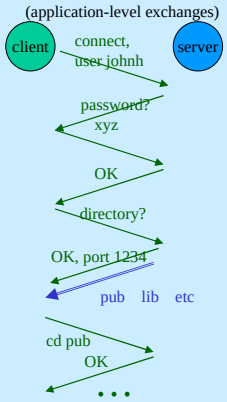
CSci551: Computer Networks
SP2006 Thursday Section
John Heidemann

Key ideas

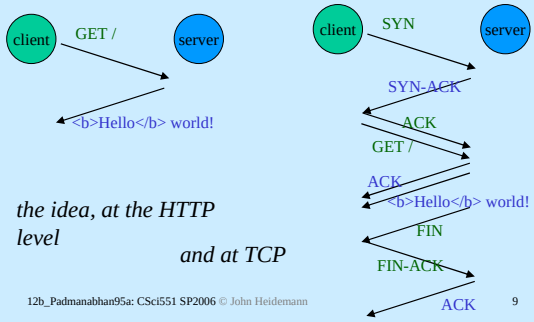
- improving HTTP latency
- performance problems (the old way)
  - many connections
    - servers have to keep track of lots of state per connects
  - TCP setup costs (3wh)
  - [authentication]
- their solution
  - persistent connections: reuse the same connection across multiple requests

FTP Review

- FTP uses two TCP connections
  - control: send commands
  - data: retrieve data (directory listings, files, etc.)
- HTTP motivation: just use one connection
  - goal: avoid server-side state
  - use a simpler protocol
- (Digression: what about this protocol is NAT unfriendly?)



HTTP Basics



the idea, at the HTTP level

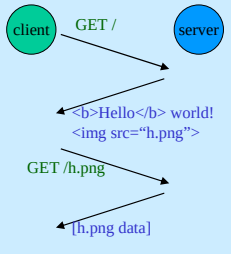
and at TCP

Web Page Content

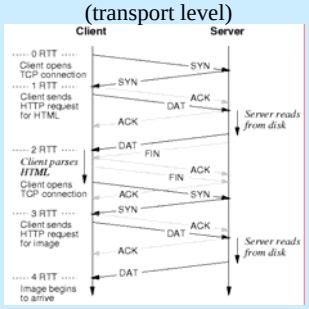
- But web pages consist of *multiple* objects
  - HTML
  - images
  - (also maybe Java, or CSS, or other things)
- HTTP/1.0: each object is a separate HTTP request, and so is a separate TCP connection

HTTP/1.0 in Practice

(application level)



(transport level)



[Padmanabhan95a,figure 3-1]

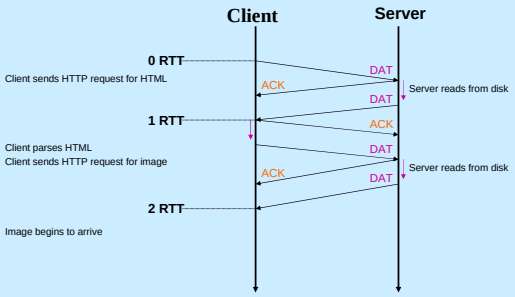
What are the problems here?

- TCP/HTTP costs
  - allocating resources (ex: TCB)
  - connection establishment
    - 3wh takes 2 RTTs
    - slow start can be a big deal with short connections
    - propgation delay itself can't be changed
    - but we can
      - reduce the number of interactions
      - cache stuff in proxy server (RTT to local proxy less than RTT to distant server)
      - or the server could place replicas around the globe (again, hopefully closer to you)
  - state left after connection tear-down

Proposed Solutions

- Persistent connections (P-HTTP):
  - use *one TCP* connection for *multiple HTTP* request/responses
  - in HTTP/1.1
- Pipelining:
  - don't wait for complete response before sending next request
  - try to send multiple requests at same time (in same packet)
  - suggestions: GETALL, GETLIST
    - not standardized

Persistent Connection Example



P-HTTP Questions

- How to multiplex requests?
  - how to distinguish multiple requests and multiple responses from each other in the TCP bytestream
  - if pipelining, need to keep track of which response belongs to which request
  - indicating end of file
    - close the connection
    - provide the length of the file (but must know beforehand the file length)
    - end-of-file delimiter (but doesn't work—no easy way to separate delimiter from data)
    - chunking

Complement: Parallel Connections

- Open *multiple* TCP connections (4)
  - first done in Netscape
- pros:
  - faster: get to overlap connection establishment, slow start
- cons:
  - still have state problem
  - doesn't help with large objects
  - interaction with congestion control: 4 connections get more bandwidth, also is hard on the network, no sharing in learning about congestion

Pipelining

- Serialized requests still have extra RTTs
- Pipelining requests
  - GETALL – request HTML document and all embeds
    - Requires server to parse HTML files
    - Doesn't consider client cached documents
  - GETLIST – request a set of documents
    - Implemented as a simple set of GETs
    - (but which ones to list?)
  - http/1.1 intstead just recommends issuing multiple requests for individual objects, but without waiting
- Also could *prefetching*
  - but risk getting data you may not need

Persistent Connection Performance

- Benefits greatest for small objects
  - Up to 2x improvement in response time
  - why are benefits for large objects no more than for medium size objects?
    - cost of startup is amortized over a larger connection
- Server resource utilization reduced due to fewer connection establishments and fewer active connections
- TCP behavior improved
  - Longer connections help adaptation to available bandwidth
  - Larger congestion window improves loss recovery

Performance Results

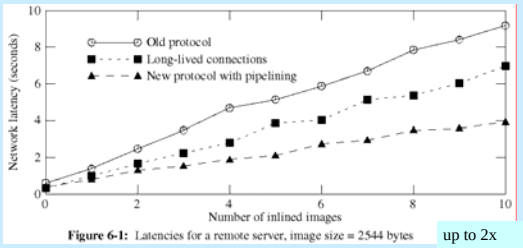


Figure 6-1: Latencies for a remote server, image size = 2544 bytes

RTT ~70ms, bottleneck bandwidth: 1.5Mb/s

up to 2x w/*10* images and pipelining

Other questions/observations?

- is this standard?
  - persistent connections and pipelining are in HTTP/1.2
  - not GETALL, etc.
- what about http/1.0 or /0.9
  - why one connection per object?
    - guess: inspired by FTP, much simplier
  - why not UDP for requests?
    - probably TCP reliability