The Application Layer – HTTP and FTP

File Transfer Protocol (FTP)

- Allows a user to copy files to/from remote hosts
  - Client program connects to FTP server
  - … provides a login id and password
  - … allows the user to explore the directories
  - … and download and upload files with the server
- A predecessor of the Web (RFC 959 in 1985)
  - Requires user to know the name of the server machine
  - … and have an account on the machine
  - … and find the directory where the files are stored
  - … and know whether the file is text or binary
  - … and know what tool to run to render and edit the file
- That is, no URL, hypertext, and helper applications

Why Study FTP?

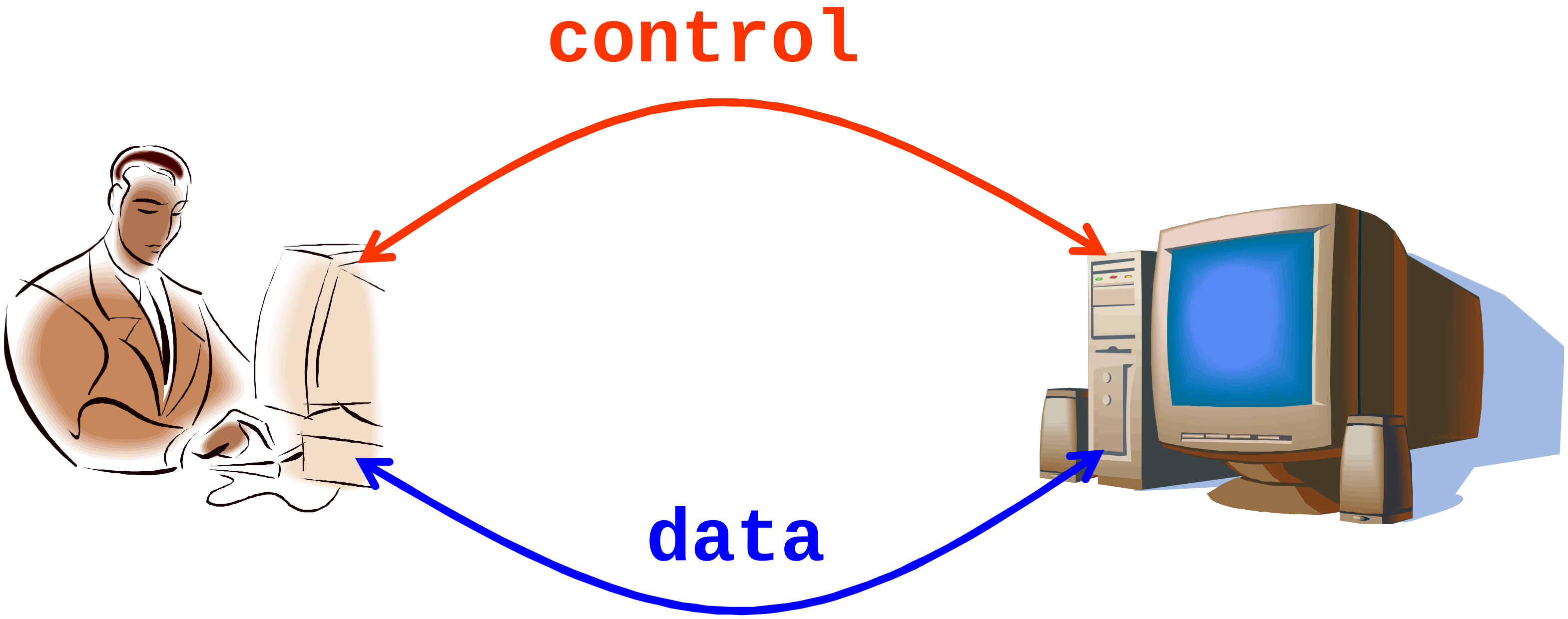
- Helps cement familiarity with text/status-code based protocols
- Illustrates use of multiple concurrent connections
  - One for control (commands & replies)
  - Depending on command, can be additional one for data
- Illustrates reversal of roles
  - For data connection, FTP user's process can play the server role, FTP server can play the client role

Example commands

- Authentication
  - USER: specify the user name to log in as
  - PASS: specify the user’s password
- Exploring the files
  - LIST: list the files for the given file specification
  - CWD: change to the given directory
- Downloading and uploading files
  - TYPE: set type to ASCII (A) or binary image (I)
  - RETR: retrieve the given file
  - STOR: upload the given file
- Closing the connection
  - QUIT: close the FTP connection

FTP Data Transfer

- Separate data connection
  - To send lists of files (LIST)
  - To retrieve a file (RETR)
  - To upload a file (STOR)

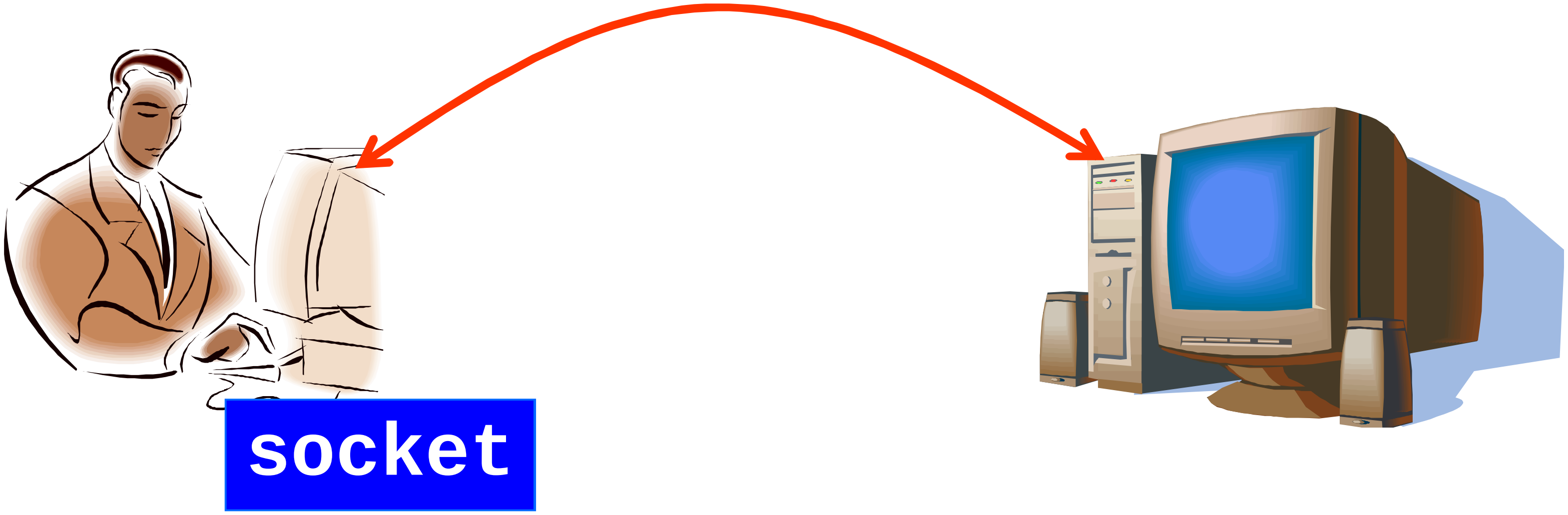


control

data

Creating the Data Connection

- Client acts like a server
  - Creates a socket
    - Assigned an ephemeral port number by the kernel
  - Listens on socket
  - Waits to hear from FTP server



Creating Data Connection (cont.)

- But, the server doesn’t know the port number
  - So after starting to listen, client tells it to the server
  - Using the PORT command on the control connection
  - Server can tell the client a port to connect to using PASV or EPSV

PORT <IP address, port #>

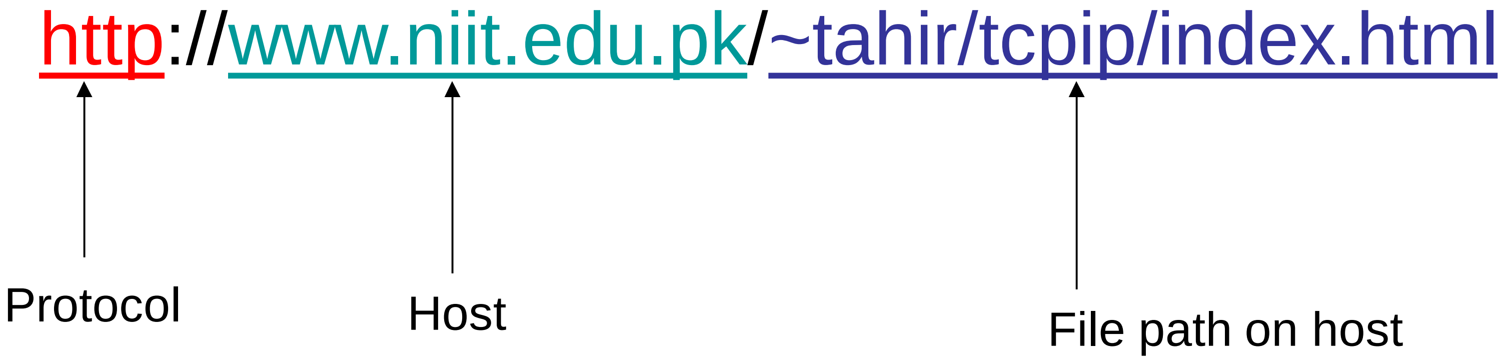
Why Out-of-Band Control?

- Avoids need to mark the end of the data transfer
  - Data transfer ends by closing of data connection
  - Yet, the control connection stays up
- Aborting a data transfer
  - Can abort a transfer without killing the control connection
  - … which avoids requiring the user to log in again
  - Done with an ABOR on the control connection
- Third-party file transfer between two hosts
  - Data connection could go to a different host
  - … by sending a different client IP address to the server
  - e.g., a user can coordinate a transfer between two servers
  - But: this is rarely needed, and presents security issues

HTTP

- Server listens on a port (by default, 80)
- On connection, waits for a request
- Protocol (but not data) is in ASCII
- Sends response, maybe closes connection (client can ask it to stay open)

Parsing a URL



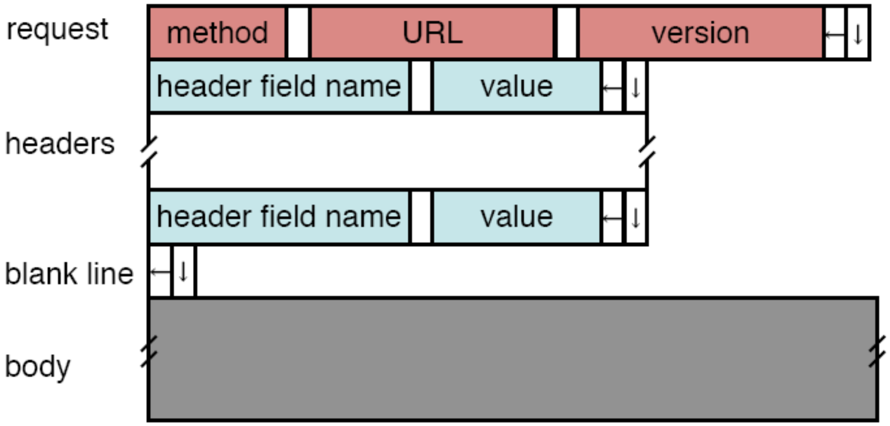
http://www.niit.edu.pk/~tahir/tcpip/index.html

Protocol

Host

File path on host

HTTP Request Format



- Request types: GET, PUT, POST, HEAD, DELETE
- A trivial browser request: http://localhost:8000

Other useful header fields

- Range: Request a partial range of data
- Authorization: Present authorization credentials to a server (not HTTPS)
- Proxy-Authorization: Present proxy credentials to a proxy server
- Referer: URL of the web page the user was on, when the HTTP request was made

HTTP Performance

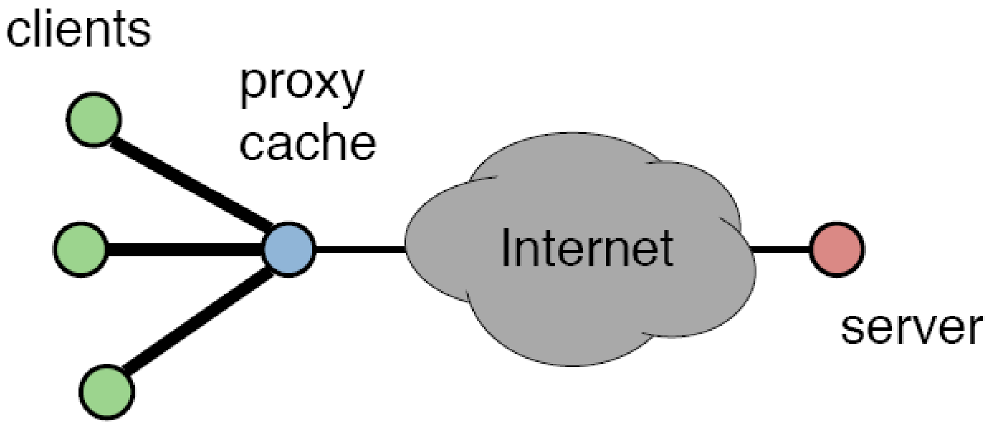
- What matters most?
- Different kinds of requests
  - Lots of small requests (loading a web page)
  - Big request (fetching a download)
- Require different solutions

Small requests

- Latency matters
- Governed by RTT between hosts
- Two major causes of delay:
  - Opening a TCP connection
  - Data response-request
- Solutions:
  - Persistent connections
  - Pre-fetching
  - Others??

Big requests

- Problem is throughput on bottleneck links (usually edge links)
- Use an HTTP proxy cache or mirror
  - Can also improve latency!



Client-server vs Peer-to-peer

- Server can be a bottleneck
  - Download time can scale down *O(n)* with *n* clients
  - Scaling up server bandwidth can be expensive (CDNs)
  - Slashdotting/flash crowds
- Peer-to-peer: get a bunch of end-hosts to collaboratively distribute content
- A common peer-to-peer challenge is finding whom to collaborate with