CS434/534: Topics in Network Systems

Layered Architecture
Network Systems Overview; FTP; HTTP

Sept. 9, 2021

Y. Richard Yang
Computer Science Department, Yale University
Email: yry@cs.yale.edu

http://zoo.cs.yale.edu/classes/cs434/
Outline

- Admin and recap
- Bigger picture
  - Physical infrastructure
  - The layering architecture
    - Layering basic concepts
    - Implications of layered network systems
    - Layering design principle
    - The Internet layers
- Networked systems (network applications, network-based systems)
  - Overview
  - HyperText Transfer Protocol
    - Basic abstractions
Recap: The Internet (w/ Structure)
Recap: Layered Architecture for NS

- A technique to organize a network system into a **succession** of logically distinct entities, such that the service provided by one entity is **solely** based on the service provided by the previous (lower level) entity.

- **Layering basic concepts**
  - Service - says **what** a layer does
  - Interface - says **how** to access the service
  - Protocol - specifies **how** the service is implemented
Recap: SMTP as Example Protocol (RFC5321)

S: 220 mr1.its.yale.edu
C: HELO cyndra.yale.edu
S: 250 Hello cyndra.cs.yale.edu, pleased to meet you
C: MAIL FROM: <spoof@cs.yale.edu>
S: 250 spoof@cs.yale.edu... Sender ok
C: RCPT TO: <yry@yale.edu>
S: 250 yry@yale.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Date: Mon, 08 Feb 2021 11:20:27 -0500 (EST)
C: From: "Y. R. Yang" <yry@cs.yale.edu>
C: To: "Y. R. Yang" <yry@cs.yale.edu>
C: Subject: This is subject
C:
C: This is the message body!
C: Please don’t spoof!
C:
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 mr1.its.yale.edu closing connection
Internet Standardization Process

- All standards of the Internet are published as RFC (Request for Comments)
  - e.g., the SMTP protocol is specified in RFC821-5321
- A typical (but not the only) way of standardization:
  - Internet draft
  - RFC
  - proposed standard
  - draft standard (requires 2 working implementations)
  - Internet standard (declared by Internet Architecture Board)
- David Clark, 1992:
  - We reject: kings, presidents, and voting. We believe in: rough consensus and running code.
Overall SMTP Protocol Framework

RFC5321 SMTP: protocol for exchanging email msgs

RFC 5322 (<-2822<-RFC822): standard for text message format:
- Header lines, e.g.,
  - To:
  - From:
  - Subject:
- Body
  - the “message”, ASCII characters only
Message Body Format: Multimedia Extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- Additional lines in msg header declare MIME content type

<table>
<thead>
<tr>
<th>MIME version</th>
<th>multimedia data type, subtype, parameter declaration</th>
<th>method used to encode data</th>
<th>encoded data</th>
</tr>
</thead>
<tbody>
<tr>
<td>From: <a href="mailto:yry@cs.yale.edu">yry@cs.yale.edu</a></td>
<td>To: <a href="mailto:cs433@cs.yale.edu">cs433@cs.yale.edu</a></td>
<td>Subject: Network map.</td>
<td>MIME-Version: 1.0</td>
</tr>
<tr>
<td>Content-Type: image/jpeg</td>
<td>Content-Transfer-Encoding: base64</td>
<td>base64 encoded data .....</td>
<td>..........................</td>
</tr>
</tbody>
</table>

Benefit of MIME type: self describing data type, adding extensibility.
Hi,
Attached is network topology map.

--98766789
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data ..... 
..........................
......base64 encoded data
--98766789--
Example SMTP w/ MIME

User agent connects to mail server via SMTP protocol:
- HELO command to identify client
- MAIL FROM:...Sender ok
- RCPT TO: ...Recipient ok
- Data transfer:
  - Date, From, To, Subject
  - Multipart/mixed; boundary=98766789
  - Text/plain
  - Base64 encoded image
- Message accepted for delivery
- QUIT command to terminate connection
(Offline) POP3 Protocol: Mail Retrieval

Authorization phase
- client commands:
  - user: declare username
  - pass: password
- server responses
  - +OK
  - -ERR

Transaction phase, client:
- list: list message numbers
- retr: retrieve message by number
- dele: delete
- quit

```
% openssl s_client -connect pop.gmail.com:995
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```
(Offline) Exercise

- Send email to your gmail with attachment
- Retrieve using pop3
Outline

- Admin and recap
- Big picture
  - Physical infrastructure
  - The layering architecture
    - Layering basic concepts
    - Implications of layered network systems
Layering Abstraction: *Logical Communication*

**E.g.: transport**
- Trans. msg for app

**Transport protocol**
- Send transport messages to transport peer, e.g.,
  - add control info such as sequence # to message; send message to peer; wait for peer to ack receipt; if no ack, retransmit
Layering Abstraction -> **Logical Communication**

E.g.: application

- provide services to users

- application protocol:
  - send application messages to application peer, e.g.,
    - email transport (SMTP) protocol sends HELO, MAIL FROM, RCPT TO between two SMTP peers
Layering Realization: *Physical Communication*
Meta Data in Layered Network System

- Each layer adds meta-data (packet header) to its payload
- Each non-highest layer should include a **multiplexing** field to indicate the higher-layer entity requiring the service

Ln

Ln’

Ln-1 (e.g., transport) header should have a field to indicate higher layer entity (e.g., which app) requiring the service
Demo: SMTP in Layered Internet

- Try smtp command as before
- Use wireshark to capture all traffic
  - Display filter: smtp
  - Display filter: tcp.port == 25
SMTP Behind the Scene

Host A
ACK(seq=x), SYN(seq=y)
ACK(seq=y)
ACK

Host B
SYN(seq=x)

Setup protocol

220 ESMT
ACK

Close protocol

FIN
ACK
FIN
ACK
Meta-data (packet headers) processing in a layering system can be considered managed as a stack.
Outline

- Admin and recap
- Big picture
  - Physical infrastructure
  - The layering architecture
    - Layering basic concepts
    - Implications of layered network systems
    - Layering design principle
The Layering Design Issue

- Key design issue: How do you divide functionalities among the layers?

- Key principle
  - “The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication systems. Therefore, providing that questioned function as a feature of the communications systems itself is not possible.” -- J. Saltzer, D. Reed, and D. Clark, 1984
Exercise: Network or Endhost

- Consider a 2-layer design (L1 placed on all nodes called the network layer, L2 end hosts only)
- Consider reliable transport as a service.
  - Design 1: L1 implements reliability: Entities in L2 share a reliable transport connection provided L1
  - Design 2: each entity in L2 implements its own reliable transport
- Discussion: benefits of putting reliability in L1 (L2)
Some Reasons Supporting Design 1

- Design 1 may improve performance, e.g., if high cost/delay/... on a local link
  - improves efficiency
  - reduces delay

- Design 1 allows sharing common code, e.g., reliability is required by multiple applications
Some Reasons Supporting Design 2

- Design 1 cannot completely provide the functionality
  - the receiver has to do the check anyway!
- Design 1 increases complexity, cost and overhead for those who may not need it
  - shared by all upper layer applications → everyone pays for it, even if you do not need it
- Design 2 can do better than Design 1
  - knows specific requirements and thus may choose specific reliability approaches
  - design 1 may create head of line blocking if not careful
A Typical Summary of End-to-End Arguments

- If a higher layer can do it, don’t do it at a lower layer -- the higher the layer, the more it knows about the best what it needs.

- Add functionality in lower layers iff it
  1. is used by and improves performance of a large number of (current and potential future) applications,
  2. does not hurt (too much) other applications, and
  3. does not increase (too much) complexity/overhead

- Practical tradeoff, e.g.,
  - allow multiple interfaces at a lower layer (one provides the function; one does not)
We used reliability as an example

Assume two layers (L1 at all nodes; L2 at end hosts). Where may you implement the following functions?
- security (privacy of traffic)
- quality of service (e.g., delay/bandwidth guarantee)
- congestion control (e.g., not to overwhelm network links or receiver)
Outline

- Admin and recap
- Big picture
  - Physical infrastructure
  - The layering architecture
    - Layering basic concepts
    - Implications of layered network systems
    - Layering design principle
    - The Internet layers
Example Internet Layering for HTTP (Before HTTP/3)
The “Hourglass” Model

Network infrastructure

end users

network infrastructure

Email HTTP FTP Telnet

TCP UDP

IP4/6

Ethernet Wireless Cable/DSL
Discussion: Limitations of Layering Architecture for Network Systems
Outline

- Admin and recap
- Big picture
- Networked systems (network applications, network-based systems)
  - overview
Network Application and Application-layer Protocol

Network application implementation (application): communicating, distributed processes
  o a process is a program that is running within a host
  o processes communicate by network application protocol(s)

Network application protocol(s)
  o one “piece” of a system
  o define messages exchanged by processes and actions taken

We study both implementation and protocol.
Commonly used transport-layer service model and API: Socket

- sometimes called "Berkeley sockets" acknowledging their heritage from Berkeley Unix
- a socket has a transport-layer local port number
  - e.g., email (SMTP) port number 25, web port number 80
- an application process uses socket system call to bind to a socket
  - `%netstat -a -p tcp -n`
  - `%lsact -i -n`
- Application can send data into socket, read data out of socket
Socket Service Model and API

buffers, states

transport protocol

buffers, states

Controlled by application developer

Controlled by operating system

Host or server

Host or server
Multiplexing/Demultiplexing
Networked Systems Study: Key Questions

Key questions we ask about a network system
- What is the architecture (entities/processes, protocols involved)?

- How does it provide
  - Extensibility
  - Scalability
  - Robustness to failures
  - Security?
Patterns of Networked Systems

- Many architectural patterns to design networked systems
  - client-server
  - peer-to-peer
  - pub/sub
  - ...

- First focus on client-server patterns, e.g.,
  - Remote procedure call (RPC)
    - will cover details later
  - HTTP
    - our focus
  - FTP
    - alternative
The FTP File Transfer Architecture

- **Service**: Transfer files to/from remote host—others such as NFS, Dropbox, ...

- **Protocol**
  - ftp: RFC 959
    - ftp server: port 21/20 (smtp 25, http 80)

  "Bragging opportunity": How may you design basic FTP protocol?
FTP Protocol, Commands, Responses

Sample commands:
Sent as ASCII text over control channel
- USER username
- PASS password
- PWD returns current dir
- STAT shows server status
- LIST returns list of file in current directory
- RETR filename retrieves (gets) file
- STOR filename stores file

Sample return codes
Status code and phrase
- 331 Username OK, password required
- 125 data connection already open; transfer starting
- 425 Can’t open data connection
- 452 Error writing file
FTP Exercise

- **Manual**
  - `ftp ftp.freebsd.org` `ftp`
  - `cd` to `pub/FreeBSD`
  - Download `README.TXT` (first use then `cd` to root and full path)

- **Use Wireshark to capture FTP traffic**

- **Details of FTP protocol see offline-read slides at the end**
Discussion

- What design decisions of FTP might have limited FTP's broader use?
Outline

- Admin and recap
- Bigger picture
- Networked systems (network applications, network-based systems)
  - Overview
  - HyperText Transfer Protocol
    - FTP as comparison
    - Basic abstractions
HTTP

- One of the most widely deployed networked systems
- Focus on HTTP/1.1 now
- We will cover HTTP/2 and HTTP/3 later in the semester
HTTP Architecture Overview

- The HTTP protocol is designed to support a generic interface protocol supporting the Representational State Transfer (REST) architecture style
  - Stateless (vs stateful FTP)
  - Universal resource (vs only files in FTP)
  - State transfer as universal operators to manipulate resources (vs diverse types of methods in RPC)
  - Self-descriptive messages allowing flexible, extensive representation
HTTP Basic Concept: Resource

- HTTP resource identification: Universal Resource Identifier (URI)

  http-URI = "http:" "//" authority path-abempty [ "?" query ] [ "#" fragment ]

- A resource is basically anything that can be named, e.g., static pictures, real-time feeds with stock prices, advanced business processes.

- Abstract URI hides the details of how a service is implemented, and hence presenting a uniform interface to clients that is independent of the types of resources provided.
HTTP Basic Concept: State Transfer

- A small number of HTTP methods to indicate state transfer (between whom?) of a resource, providing the other component of a uniform interface

  - **GET**
    - server -> client; safe, idempotent

  - **PUT**
    - client -> server; idempotent

  - **POST**
    - client -> server; not idempotent

  - **DELETE**

  - **HEAD**
  - **CONNECT**
  - **OPTION**
  - **TRACE**
Exercise: “Conforming” RESTful Use of HTTP Methods

<table>
<thead>
<tr>
<th>Task</th>
<th>Method</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a new customer</td>
<td></td>
<td>/customers</td>
</tr>
<tr>
<td>Delete an existing customer</td>
<td></td>
<td>/customers/{id}</td>
</tr>
<tr>
<td>Get a specific customer</td>
<td></td>
<td>/customers/{id}</td>
</tr>
<tr>
<td>Search for customers</td>
<td></td>
<td>/customers</td>
</tr>
<tr>
<td>Update an existing customer</td>
<td></td>
<td>/customers/{id}</td>
</tr>
</tbody>
</table>
(Offline) Exercise: “Non-Conforming” REST Example

- Twitter “Direct Message” REST API

  ```
  GET /direct_messages/show.json?id={id}
  POST /direct_messages/destroy.json?id={id}
  POST /direct_messages/new.json
  ```

- Exercise: How may “pure” REST API look like?

  ```
  GET /direct-messages/{id}
  
  DELETE /direct-messages/{id}
  
  POST /direct-messages
  ```
HTTP/1.x Request Message: General Format

- ASCII (human-readable format)

```
HTTP/1.x Request Message: General Format

- ASCII (human-readable format)

```

```
method sp URL sp version cr lf
header field name : value cr lf
header lines
header field name : value cr lf
request line
Entity Body
```
Demos

- Using curl (curl -v -X <method> --http<ver> -H <header> <uri>)
  - curl -v http://www.google.com
  - curl -v -X OPTIONS http://www.google.com
HTTP Response Message Example

- **status line**
  - (protocol
  - status code
  - status phrase)

- **HTTP/1.0 200 OK**

- **header lines**
  - Date: Wed, 23 Jan 2008 12:00:15 GMT
  - Server: Apache/1.3.0 (Unix)
  - Last-Modified: Mon, 22 Jun 1998 ......
  - Content-Length: 6821
  - Content-Type: text/html

- **data, e.g., requested html file**
  - data data data data data data data ...
HTTP/1.x Response Status Codes

In the first line of the server->client response message. A few sample codes:

200 OK
  o request succeeded, requested object later in this message

301 Moved Permanently
  o requested object moved, new location specified later in this message (Location:)

400 Bad Request
  o request message not understood by server

404 Not Found
  o requested document not found on this server

505 HTTP Version Not Supported
Offline HTTP: POST Request Example

POST /path/script.cgi HTTP/1.0
User-Agent: MyAgent
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

item1=A&item2=B
Outline

- Admin and recap
- Bigger picture
- Networked systems (network applications, network-based systems)
  - Overview
  - HyperText Transfer Protocol
    - FTP as comparison
    - Basic abstractions
      - Resource
      - State transfer
      - Representation
HTTP Basic Concept: Representations

- A resource can have multiple representations
- HTTP methods transfer representations of a resource
- Exercise: What are some concrete examples of multiple representations of a single resource?
HTTP Basic Concept: Content (Representation) Negotiation

- HTTP dimensions to determine the representation
  - Client characteristics
  - Negotiation fields
    - Accept (mime type), e.g., text, html, jpeg, png
    - Accept-Charset, e.g., ascii, ...
    - Accept-Encoding, e.g., gzip, compress
    - Accept-Language, e.g., en

- Discussion: what may be potential designs to choose a representation when a client retrieves a resource?
Two Types of (Content) Negotiation in HTTP

- **Proactive content negotiation (aka server-driven)**
  - When content negotiation preferences are sent by the user agent in a request to encourage an algorithm located at the server to select the preferred representation
    - Client indicates in attributes (e.g., User-Agent field) and negotiation fields (examples?)

- **Reactive content negotiation**
  - Server indicates availability in response Vary header, and client requests the preferred one
Exercise

- Using curl (curl -v -X <method> --http<ver> -H <header> <uri>
  - curl -v http://www.google.com
    find Google server indicating representations possibilities
Exercise

- Using curl (curl -v -X <method> --http<ver> -H <header> <uri>
  vs
Next Class

- Finish HTTP/1.1 additional features
- Network server design
  - Socket API
  - Thread network servers
Backup: FTP Protocol Details
Offline Exercise Puzzle

- Try
  
curl -v dus.cs.yale.edu

vs

telnet dus.cs.yale.edu 80
GET / HTTP/1.0
HTTP/1.x Request Message Example

- request line (GET, POST, HEAD, PUT, DELETE, TRACE ... commands)
- header lines
- Carriage return, line feed indicates end of message
- Virtual host multiplexing
- Connection management
- Content negotiation

```
GET /somedir/page.html HTTP/1.0
Host: www.somechool.edu
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: en

(extra carriage return, line feed)
```
FTP: A Client-Server Application with Separate Control, Data Connections

- **Two** types of TCP connections opened:
  - **A control connection**: exchange commands, responses between client, server.  
    "out of band control"
  - **Data connections**: each for file data to/from server

Discussion: why does FTP separate control/data connections?

Q: How to create a new data connection?
Traditional FTP: Client Specifies Port for Data Connection

FTP client\n
FTP server

TCP control connection\nport 21 at server

PORT clientip:cport

RETR file.dat

Server initiates TCP data connection\nserver:20\nclientip:cport
Example using telnet/nc (Offline, On Campus)

- Use telnet for the control channel
  - telnet ftp.gnu.org 21
  - user, pass
  - port 172.27.21.251,4,1 [change to Yale IP]
  - list

- use nc (NetCat) to receive/send data with server
  - nc -v -l 1025
Problem of the Client PORT Approach

- Many Internet hosts are behind NAT/firewalls that block connections initiated from outside
FTP PASV: Server Specifies Data Port, Client Initiates Connection

FTP client

TCP control connection
port 21 at server

PORT clientip:cport

RETR file.dat

Server initiates TCP data connection
server:20
clientip:cport

FTP server

FTP client

TCP control connection
port 21 at server

PASV

serverip:sport

RETR file.dat

Client initiates TCP data connection of PASV returned
serverip:sport

FTP server
Summary: FTP Protocol Design

- Two types of TCP connections between client and server:
  - A control connection: exchange commands, responses
    - Ascii command; status code + response
    - Commands may change server state
  - Data connections: each for file data to/from server

- Two ways to establish data connections

FTP client  FTP server  FTP client  FTP server

TCP control connection port 21 at server

Server initiates TCP data connection
server:20 clientip:cport

Client initiates TCP data connection of PASV returned
serverip:sport

PORT clientip:cport

RETR file.dat

PASV

serverip:sport

RETR file.dat

Client initiates TCP data connection of PASV returned