Quick PA1 Clarification

- Verification status should be "Verified", if it is "Not verified", that means you made a mistake somewhere…
Chapter 2
Application Layer

A note on the use of these Powerpoint slides:
We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we’d like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

© All material copyright 1996-2016
J.F Kurose and K.W. Ross, All Rights Reserved
Chapter 2: outline

2.1 principles of network applications
2.2 Web and HTTP
2.3 electronic mail
   • SMTP, POP3, IMAP
2.4 DNS
2.5 P2P applications
2.6 video streaming and content distribution networks
2.7 socket programming with UDP and TCP
Chapter 2: outline

2.1 principles of network applications
2.2 Web and HTTP
2.3 electronic mail
   - SMTP, POP3, IMAP
2.4 DNS
2.5 P2P applications
2.6 video streaming and content distribution networks
2.7 socket programming with UDP and TCP
Electronic mail

Three major components:
- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent
- a.k.a. “mail reader”
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server
Electronic mail: mail servers

mail servers:

- **mailbox** contains incoming messages for user
- **message queue** of outgoing (to be sent) mail messages
- **SMTP protocol** between mail servers to send email messages
  - client: sending mail server
  - “server”: receiving mail server
Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCII
Scenario: Alice sends message to Bob

1) Alice uses UA to compose message “to” bob@someschool.edu
2) Alice’s UA sends message to her mail server; message placed in message queue
3) Client side of SMTP opens TCP connection with Bob’s mail server
4) SMTP client sends Alice’s message over the TCP connection
5) Bob’s mail server places the message in Bob’s mailbox
6) Bob invokes his user agent to read message

Application Layer 2-8
Sample SMTP interaction

S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
Try SMTP interaction for yourself:

- `telnet servername 25`
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)
SMTP Wireshark Trace

- blat-filtered-trace.pcap
SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF. CRLF to determine end of message

comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response message
- SMTP: multiple objects sent in multipart message
Mail message format

SMTP: protocol for exchanging email messages
RFC 822: standard for text message format:
  - header lines, e.g.,
    - To:
    - From:
    - Subject:
      different from SMTP MAIL FROM, RCPT TO: commands!
  - Body: the “message”
    - ASCII characters only
Mail access protocols

- **SMTP**: delivery/storage to receiver’s server
- mail access protocol: retrieval from server
  - **POP**: Post Office Protocol [RFC 1939]: authorization, download
  - **IMAP**: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored messages on server
  - **HTTP**: gmail, Hotmail, Yahoo! Mail, etc.
**POP3 protocol**

*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**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on

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
```
**POP3 (more) and IMAP**

**more about POP3**
- previous example uses POP3 “download and delete” mode
  - Bob cannot re-read e-mail if he changes client
- POP3 “download-and-keep”: copies of messages on different clients
- POP3 is stateless across sessions

**IMAP**
- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name
IMAP

Borrowed from
CSCI-4220 Network Programming
Spring 2017
E-mail is Simple

- **Server**
  - Get message
  - Put in right mailbox
  - Send to client

- **Client**
  - Get message from server/give message to server
POP (briefly)

- POP, or Post Office Protocol
- Several versions, POP3 still used
- Less complicated than IMAP
- A lot more constraints on usage
- Not today’s topic!
IMAP

- Various names/incarnations, but currently IMAP4 (RFC3501)
- “Internet Message Access Protocol”
- Mailbox session and management
Core Features

- IMAP supports multiple concurrent users in a mailbox
- Sessions instead of authenticate-send/receive-disconnect pattern (POP3)
- Designed for Retrieval
- Multiple Mailboxes
- Server-side operations e.g. search
Some Design Points

- IMAP servers and clients communicate a fair amount
- We’ll examine these requests in a moment
- IMAP client may save things, considered a local cache by the server
- Concurrency means server/clients must handle unexpected state changes
- Server can in-band send unilateral data
Getting Stranger…

- 7-bit ASCII
  - What does this give us? What doesn’t it?
- What happens when we want to use special characters?
- Other encodings like UTF-8 (RFC 1648, RFC 5738, RFC 6855)
  - Yes there is a UTF-7, yes it does get used for things like international mailbox names
- CRLF terminates lines, not NUL
MIME?

- Multipurpose Internet Mail Extensions
- "MIME types" are more commonly discussed, but MIME is a lot more
  - Wunderlist required a MIME content type in the header of application/json
- RFC 1341, RFC 1521-1524, RFC 2045-2049… etc.
- Representing message contents is non trivial!
Command Types

- Defaults: TCP Port 143 for unencrypted, Port 993 otherwise
- Connection
- Server Greeting
- Server/Client Commands
Client/Server

- Client makes a tag
  - Wiki example is A001, etc.
- If server needs more info it will send a +, otherwise it echoes back the tag with response
- The server can send messages that aren’t a response/continuation by starting with *
- This is called an untagged message
Server Responses

- BAD with tag if syntax is wrong
- NO if the command failed
- OK if the command succeeded
Message Attributes

- **Unique Identifier (UID)**
  - 32 bits, MUST NOT change during session, SHOULD NOT change between sessions
  - Strictly ascending

- **Unique Identifier Validity**

- **Sequence Number**
  - Relative Positioning

- **Flags**
Logging Out

- C: 9404 LOGOUT
- S: * BYE
- S: 9404 OK

- If a client didn’t LOGOUT but disconnected, server can skip the BYE
- Clicking X on Thunderbird didn’t skip the LOGOUT!
Wireshark

- Let’s look at an example capture.
Disclaimer

- This is just a high level discussion of IMAP. If you think it still sounds simple, glance over Sections 6 & 7 of RFC3501 and think about implementing each command...

- Also check out Section 8 for ABNF (Augmented BNF) if you like formal syntaxes.
Chapter 2: outline

2.1 principles of network applications
2.2 Web and HTTP
2.3 electronic mail
   • SMTP, POP3, IMAP
2.4 DNS
2.5 P2P applications
2.6 video streaming and content distribution networks (CDNs)
2.7 socket programming with UDP and TCP
Video Streaming and CDNs: context

- **Video traffic**: major consumer of Internet bandwidth
  - Netflix, YouTube: 37%, 16% of downstream residential ISP traffic
  - ~1B YouTube users, ~75M Netflix users

- **Challenge**: scale - how to reach ~1B users?
  - single mega-video server won’t work (why?)

- **Challenge**: heterogeneity
  - different users have different capabilities (e.g., wired versus mobile; bandwidth rich versus bandwidth poor)

- **Solution**: distributed, application-level infrastructure
Multimedia: video

- video: sequence of images displayed at constant rate
  - e.g., 24 images/sec
- digital image: array of pixels
  - each pixel represented by bits
- coding: use redundancy within and between images to decrease # bits used to encode image
  - spatial (within image)
  - temporal (from one image to next)

**Spatial Coding Example:** instead of sending \( N \) values of same color (all purple), send only two values: color value (purple) and number of repeated values (\( N \))

**Temporal Coding Example:** instead of sending complete frame at \( i+1 \), send only differences from frame \( i \)
Multimedia: video

- **CBR: (constant bit rate):** video encoding rate fixed
- **VBR: (variable bit rate):** video encoding rate changes as amount of spatial, temporal coding changes
- **examples:**
  - MPEG 1 (CD-ROM) 1.5 Mbps
  - MPEG 2 (DVD) 3-6 Mbps
  - MPEG 4 (often used in Internet, < 1 Mbps)

**Spatial coding example:** instead of sending $N$ values of the same color (all purple), send only two values: color value (purple) and number of repeated values ($N$)

**Temporal coding example:** instead of sending complete frame at $i+1$, send only differences from frame $i$
Streaming stored video:

simple scenario:

video server
(stored video)

Internet

client
Streaming multimedia: DASH

- **DASH:** Dynamic, Adaptive Streaming over HTTP

  - **server:**
    - divides video file into multiple chunks
    - each chunk stored, encoded at different rates
    - *manifest file:* provides URLs for different chunks

  - **client:**
    - periodically measures server-to-client bandwidth
    - consulting manifest, requests one chunk at a time
      - chooses maximum coding rate sustainable given current bandwidth
    - can choose different coding rates at different points in time (depending on available bandwidth at time)
Streaming multimedia: DASH

- **DASH**: Dynamic, Adaptive Streaming over HTTP
- “intelligence” at client: client determines
  - *when* to request chunk (so that buffer starvation, or overflow does not occur)
  - *what encoding rate* to request (higher quality when more bandwidth available)
  - *where* to request chunk (can request from URL server that is “close” to client or has high available bandwidth)
Content distribution networks

- **Challenge**: how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

- **Option 1**: single, large “mega-server”
  - single point of failure
  - point of network congestion
  - long path to distant clients
  - multiple copies of video sent over outgoing link

….quite simply: this solution *doesn’t scale*
Content distribution networks

- **challenge:** how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?

- **option 2:** store/serve multiple copies of videos at multiple geographically distributed sites (CDN)
  - *enter deep:* push CDN servers deep into many access networks
    - close to users
    - used by Akamai, 1700 locations
  - *bring home:* smaller number (10’s) of larger clusters in POPs near (but not within) access networks
    - used by Limelight
Content Distribution Networks (CDNs)

- CDN: stores copies of content at CDN nodes
  - e.g. Netflix stores copies of MadMen
- subscriber requests content from CDN
  - directed to nearby copy, retrieves content
  - may choose different copy if network path congested
Content Distribution Networks (CDNs)

"over the top"

Internet host-host-host communication as a service

OTT challenges: coping with a congested Internet
- from which CDN node to retrieve content?
- viewer behavior in presence of congestion?
- what content to place in which CDN node?

more .. in chapter 7
CDN content access: a closer look

Bob (client) requests video http://netcinema.com/6Y7B23V
- video stored in CDN at http://KingCDN.com/NetC6y&B23V

2. Resolve http://netcinema.com/6Y7B23V via Bob’s local DNS
3. netcinema’s DNS returns URL http://KingCDN.com/NetC6y&B23V
4&5. Resolve http://KingCDN.com/NetC6y&B23V via KingCDN’s authoritative DNS, which returns IP address of KingCDN server with video
6. Request video from KINGCDN server, streamed via HTTP

Application Layer 2-43
1. Bob manages Netflix account
   - Netflix registration, accounting servers

2. Bob browses Netflix video
   - Amazon cloud
   - CDN server
   - Manifest file returned for requested video

3. Manifest file returned for requested video
   - CDN server
   - Upload copies of multiple versions of video to CDN servers

4. DASH streaming
   - Application Layer 2-44