Principles of Network Applications
Chapter 2
Application Layer

A note on the use of these ppt 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 1996-2012
J.F Kurose and K.W. Ross, All Rights Reserved
Some network apps

- e-mail
- web
- text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video
- (YouTube, Hulu, Netflix)
- voice over IP (e.g., Skype)
- real-time video conferencing
- social networking
- search
- …
- …
Creating a network app

write programs that:
run on (different) end systems
communicate over network
e.g., web server software
communicates with browser software

no need to write software for
network-core devices
network-core devices do not run
user applications
applications on end systems allows
for rapid app development, propagation
Application architectures

possible structure of applications: client-server peer-to-peer (P2P)
Client-server architecture

**server:**
- always-on host
- permanent IP address
- data centers for scaling

**clients:**
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other
**P2P architecture**

no always-on server

arbitrary end systems directly communicate

peers request service from other peers, provide service in return to other peers

*self scalability* – new peers bring new service capacity, as well as new service demands

peers are intermittently connected and change IP addresses

complex management
Processes communicating

**process:** program running within a host
within same host, two processes communicate using inter-process communication (defined by OS)

**client process:** process that initiates communication

**server process:** process that waits to be contacted

✦ aside: applications with P2P architectures have client processes & server processes
Sockets

process sends/receives messages to/from its socket
socket analogous to door
sending process shoves message out door
sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process

controlled by app developer
controlled by OS

Application Layer
Addressing processes

to receive messages, process must have *identifier*
host device has unique 32-bit IP address

**Q:** does IP address of host on which process runs suffice for identifying the process?

- **A:**

  *identifier* includes both IP address and port numbers associated with process on host.

  example port numbers:
  - HTTP server: 80
  - mail server: 25

  to send HTTP message to gaia.cs.umass.edu web server:
  - **IP address:** 128.119.245.12
  - **port number:** 80

  more shortly…
App-layer protocol defines

types of messages exchanged, e.g., request, response

message syntax:
what fields in messages & how fields are delineated

message semantics
meaning of information in fields

rules for when and how processes send & respond to messages

open protocols:
defined in RFCs
allows for interoperability
e.g., HTTP, SMTP

proprietary protocols:
e.g., Skype
What transport service does an app need?

**data integrity**
- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

**throughput**
- some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
- other apps (“elastic apps”) make use of whatever throughput

**timing**
- some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”

**security**
- encryption, data integrity, …
Transport service requirements: common apps

<table>
<thead>
<tr>
<th>application</th>
<th>data loss</th>
<th>throughput</th>
<th>time sensitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>file transfer</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>e-mail</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>Web documents</td>
<td>no loss</td>
<td>elastic</td>
<td>no</td>
</tr>
<tr>
<td>real-time audio/video</td>
<td>loss-tolerant</td>
<td>audio: 5kbps-1Mbps, video: 10kbps-5Mbps</td>
<td>yes, 100’s msec</td>
</tr>
<tr>
<td>stored audio/video</td>
<td>loss-tolerant</td>
<td>same as above</td>
<td></td>
</tr>
<tr>
<td>interactive games</td>
<td>loss-tolerant</td>
<td>few kbps up</td>
<td>yes, few secs</td>
</tr>
<tr>
<td>text messaging</td>
<td>no loss</td>
<td>elastic</td>
<td>yes, 100’s msec</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>yes and no</td>
</tr>
</tbody>
</table>
Internet transport protocols services

TCP service:
reliable transport between sending and receiving process
flow control: sender won’t overwhelm receiver
congestion control: throttle sender when network overloaded

does not provide: timing, minimum throughput guarantee, security
connection-oriented: setup required between client and server processes

UDP service:
unreliable data transfer between sending and receiving process
does not provide: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup,

Q: why bother? Why is there a UDP?
# Internet apps: application, transport protocols

<table>
<thead>
<tr>
<th>application</th>
<th>application layer protocol</th>
<th>underlying transport protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>e-mail</td>
<td>SMTP [RFC 2821]</td>
<td>TCP</td>
</tr>
<tr>
<td>remote terminal access</td>
<td>Telnet [RFC 854]</td>
<td>TCP</td>
</tr>
<tr>
<td>Web</td>
<td>HTTP [RFC 2616]</td>
<td>TCP</td>
</tr>
<tr>
<td>file transfer</td>
<td>FTP [RFC 959]</td>
<td>TCP</td>
</tr>
<tr>
<td>streaming multimedia</td>
<td>HTTP (e.g., YouTube), RTP [RFC 1889]</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td>Internet telephony</td>
<td>SIP, RTP, proprietary (e.g., Skype)</td>
<td>TCP or UDP</td>
</tr>
</tbody>
</table>
Securing TCP

TCP & UDP
- no encryption
- cleartext passwds sent into socket traverse Internet in cleartext

SSL
- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer
- Apps use SSL libraries, which “talk” to TCP
- SSL socket API
- cleartext passwds sent into socket traverse Internet encrypted
- See Chapter 7

2-16