CS 456 Hao Pan
Salahuddin, Mohammad
Spring 2018

Section 2.1 Principles of Network Applications

22/06/2018 [T]

2.1.1 Example Network Apps

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

2.1.2 Creating a Network App

- To create a network app, it must:
  - Run on *different* end systems.
  - Communicate over a network.
  - **Ex.** A web server software communicates with browser software.
- There is no need to write software for *network-core devices* because:
  - Network core devices do not run user applications.
  - Applications on end systems allow for rapid app development and propoation.
- There are *two* possible structure of applications:
  - Client-server
  - Peer-to-peer (P2P)

2.1.3 Client-server Architecture

- **Servers**:
  - Are an always-on host.
  - Have a permanent IP address.
  - Have data centers for scaling.
- **Clients**:
  - Communicate with servers.
  - May or may not be intermittently connected.
  - Could have dynamic IP addresses.
  - Do not directly communicate with each other.

2.1.4 P2P Architecture

- *Not* an always-on server.
- Random end systems directly communicate.
- Peers *request* service from other peers and *provide* service to other peers in return.
- Peers are not always connected and change IP addresses. Thus management is very complex.

2.1.5 Processes Communicating

- **Process** refers to a program running within a host.
- Within the *same host*, two processes communicate using inter-process communication.
- Processes in different hosts communicate by exchanging messages.
- In the client/server model, the client process initiates communication and the server process waits to be contacted.
- **Note:** Applications with P2P architectures have both client and server processes.

2.1.6 Sockets

- Processes send or receive messages to/from their sockets.
- A **socket** can be compared to a door:
  - The sending process "shoves" a message out the door.
  - It relies on the *transport infrastructure* on the other side of the door to deliver the message to the socket at the receiving process.

2.1.7 Addressing Processes

- To receive messages, processes must have an *identifier*.
- A host device has a unique 32-bit IP address
- The host's IP address is not enough to identify the process because multiple processes can run on the same host.
- Thus, the **identifier** includes both the *IP address* and the it port numbers associated with the process on the host.
- Example port numbers:
  - HTTP server: 80
  - mail server: 25
- **Ex.** To send HTTP messages to the gaia.cs.umass.edu web server, use:
  - IP address: 128.119.245.12
  - Port number: 80

2.1.8 App-layer Protocol Defines

- Types of messages exchanged (request, response...)
- Message syntax (message fields)
- Message semantics (Meaning of info in fields)
- Rules for when and how process send/respond to messages.
- Open protocols (HTTP, SMTP...)
- Proprietary protocols (Skype..)

2.1.9 Transport Services Needed by an App

- Data Integrity
  - Some apps (such as file transfers or web transactions) require 100% *reliable* data transfers.
  - Other apps, like audio, can tolerate a bit of loss.
- Timing
  - Some apps (Internet, telephone, games...) are only *effective with low delay*.
- Throughput
  - Some apps (such as multimedia) require only a minimal amount of throughput to work effectively.
  - Some apps ("elastic apps") will use whatever throughput they can get.
- Security
  - Encryption, data integrity, etc. is important to ensure data does not get in the wrong hands.
- Note: The table on slide 2-14 shows the importance of each service for different apps.

2.1.10 Internet Transport Protocol Services

TCP Service:

- *Reliable transport* between sending and receiving processes.
- *Flow control* so the sender does not overwhelm the receiver.
- *Congestion control* throttles the sender when the network is overloaded (so data is not lost).
- **Does not provide**: Timing, minimum throughput guarantee, or security.
- *Connection oriented*, so setup is required between client and server processes.

UDP Service:

- *Unreliable data transfer* between sending and receiving processes.
- **Does not provide**: Reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

Note: The table on slide 2-16 shows whether TCP or UDP is preferred in different apps.

2.1.11 Securing TCP

- **TCP & UDP** have no encruption. Cleartext passwords sent into a socket *traverse in cleartext*.
- **SSL** provides encrypted TCP connection. SSL has data integrity and end-point authentication.
- SSL is at the app layer. Apps use SSL libraries that communicate with TCP.
- The SSL socket API encrypts passwords before sending them into the socket so they *traverse encrypted*.