Lecture 2 - Application Layer

Intro to Application layer

Creating a network Application

- the app runs on different end systems and communicate over network
- and there is no need for writing code for net work drivers and hardware

Principles of network applications

Overview

- Process → socket → network

1. **Process**
    - def: program running within a host
        - within same host, two processes communicate using inter-process communication (IPC)
        - processes in different hosts communicate by exchanging messages
2. **Socket**
    - Process sends and receives messages to / from socket
    - Process analogous to door
        - sending process **push** messages out of the door
        - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
        - 2 socket involved: one on each side
3. Addressing processes
    - process should have identifier
    - IPV4: 32-bit, IPV6: 128-bit
    - IPV4: 192.168.1.100:80, IPV6: [240e:3a1:4cb1:69d0:f40c:4269:74a2:7ea3]:80
4. Client-Server Paradigm
    (a) **Servers**
    - always-on host
    - Permanent IP address
        - Often in data centers, for scaling

    (b) **Clients**
    - Contact, commmunicate with server
    - Many be intermittenly conect to the internet
    - Many have dynamic IP address
    - Do not communicated directly with each other
5. Peer-to-Peer Paradigm
    - No always-on server is needed
    - End systems can act as both clients and servers
    - Client and server processes can run on the same host
    - self scalability
    - peers are intermittently connected
    - can join and leave the network at any time
    - no need for a central server to manage connections
    - optional dynamic IP addresses, hard to manage

transportation

service requirements

1. Data Integrity

    100% reliable data transfer or Tolerate some loss like: file transfer: 100% reliable, and no loss, throughout can be elastic, no time sensitive
    - email
    - file transfer
    - web docs
2. Timing

    some apps requires low delay like:
    - real time video / audio: no loss tolerated, low delay, throughput based on the quality
    - streaming video / audio: some loss tolerated, low delay preferred but can be tolerated for even seconds, throughput based on the quality
3. Troughout

    some apps requires minimum amount of throughput to be "effective" other not requieres
    - real time video / audio: no loss tolerated, low delay, throughput based on the quality
    - streaming video / audio: some loss tolerated, low delay preferred but can be tolerated for even seconds, throughput based on the quality
4. Security:

    requires encryption, dataintegrity check

Protocols

1. TCP
    - reliable transport
    - flow control: will not overwhelm receiver
    - Congestion control: will not overwhelm network
    - NO OFFER:
        - timing,
        - minimum throughput guaranteem
        - security
    - Connection-oriented: the clients and server's connectio requires setup
2. UDP
    - unrealiable transfer
        - does not offer: reliable, flow control, congestion control, timing, throughput gurantee, security, connnection setup
    - BUT IT IS **SIMPLE** AND **FAST**
3. Examples:
    - email { SMTP [RFC 2821], POP3, IMAP }: TCP
    - Remote terminal Access { telnet [RFC 854], SSH }: TCP
    - Web { HTTP [RFC 2616] }: TCP

- File Transfer { FTP [RFC 959], TFTP [RFC 1350] }: TCP, UDP
- Streaming Multimedia { RTP [RFC 3550] }: UDP
- DNS { RFC 1035 }: UDP, TCP

4. Securing TCP - Secure Socket Layer - SSL

   → protocol layer: TCP & UDP

   - no encryption
   - cleartext pwd → SSL: located at application layer
   - provides encrypted TCP connection
   - Provides encrypted TCP connection
   - Data integrity
   - End-point authentication

   → SSL socket API:

   - cleartext pwd → encrypted pwd → Internet

Web Application - HTTP

HTTP

- Hypertext Transfer Protocol
- Located at Application Layer
- Client / Server model
- Web Server ⟷ Clients

1. HTTP 1.1 Features

   Client ->[http port 80, https 443]-> Server Clinet <-[Accept TCP form client]<- Server Client <-[http msgs exchange]-> Server [TCP CONNECTION CLOSED]

2. general http **request** msg format

   - **request line**: method sp | url sp | version cr lf
   - **header lines**: header field name: value cr lf header field name: value cr lf header field name: value cr lf …

   - \r\n
     - **body**: entier body (human readable or binary)

   (a) Header lines:

   contains the key-value pairs metadata about the request / response

   - host name, authentication - content type - User-agent information - Caching / Cookies - Types of connection

3. General HTTP **Response** msg format

   - **request line**: version sp | STATUS CODE sp | STATUS TEXT cr lf
   - **header lines**: header field name: value cr lf header field name: value cr lf header field name: value cr lf …

   - \r\n
     - **body**: entier body (human readable or binary)

   (a) STATUS CODE:

   - 1XX: INformational like request received, continuing process
   - 2XX: Success
   - 200: OK
   - 3XX: Redirection
   - 301 Move Permanently
   - 4XX: Client Error
   - 400 Bad Request
   - 404 Not Found
   - 5XX: Server Error
   - 505 HTTP Version Not Supported

4. CRUD request methods in HTTP 1.1

   - GET
   - Create
   - Modify
   - Delete

   (a) HTTP 1.0 1.1 Addtional Methods

   - GET - HEAD
   - POST - TRACE
   - PUT - OPTIONS
   - DELETE - CONNECT
   - PATCH

5. HTTP Connection Types

   (a) None Persistent HTTP

   At most one object sent over TCP Conection Connnect → transfer one → closed TCP Downloads multiple objects → multiple TCP connections or multiple request-response pairs

   i. RTT: Round Trip Time

   - time for a small packet to travel from client to server and back

   ii. HTTP response time:

   one RTT to initiate TCP conection one RRT for http request and first few bytes of http response to return file transmission time **Overall**: 2RTT + File Transmission time

   iii. Issues:

   each object requires 2 RTT + file transmission time OS overhead for eacch TCP connection Browser often openparallel TCP connnections to fetch referenced objects

   (b) Persistent HTTP

   Multiple obj can be send over single TCP conection between client and server

   i. ISSUES / ADVANTAGES

   - the server will stay in open state after client sending the request
   - client and server can send msg throogh opened connecction
   - once the client encounters referenced objects, it can send requests immediately
   - every referenced object requires only one RTT

6. User-Server State: Cookies

   cause' the http protocol is stateless, we use the cookits to store the user-server state

   (a) Flow

   i. Cookie header line of HTTP response message
   ii. cookie header line in next HTTP request message
   iii. Cookie file kept on user's host
   iv. back-end database at Web site

   (b) Usage

   - authentication

- Recommendation
- User session state

7. Cache:

    - Cache acts as both client and server
        - server for original requesting client
        - client to origin server
    - Typically cache is installed by ISP
    - it can shorten response time for client request
    - and decrease the traffic on an institution's access link to the Internet
    - the cache also provides support for poor content (like P2P)

HTTP 2

HTTP 1.1: introduce multiple, piplined GETs over single TCP connection

- Server responds in-order (FCFS: first come first server scheduling)
- With FCFS, small object may have to wait for transmission of large object
- loss recovery (retransmitting lost TCP segments) stalls object

HTTP / 2: increaser flexibility at server in sending object to client

- methods, status codes, most header fields unchanged from HTTP 1.1
- transmission order of requested objects based on client-specified priority
- push unrequested objects to client
- devide object into frames, schedule frame to mitigate HOL blocking

goal: decrease response time and cost

1. Mitigating HOL Blocking

    (a) HTTP1.1:

    client requests 1 large object and 3 smaller obejcts server send in FCFS order

    (b) HTTP/2:

    objects devided into frames, frame transmission interleaved object1 large, O2-4 small Object 2,3,4 frames interleaved with object 1 frames and O1 deliver delayed, O2-4 delivered earlier

HTTP/3:

HTTP/2 will stall all the pkg if one pkg is lost and HTTP/3 uses QUIC over UDP to mitigate the issue

1. QUIC: Quick UDP Internet Connections

    - error and congestion control
    - coneection establishment: realiability, congestion control, authentication, encryption, state establishment in **one RTT**

2. Connection establishment

    (a) TCP + TLS:

    RTT1: TCP handshake RTT2: handshake TLS … (data)

    → Handshake in 2 RTTs

    (b) QUIC:

    only one RTT is needed

    (c) 0-RTT Connnection Establishment

    - Client user last session ticket for encryption, authentication for new connection
    - use last ticket, no handshake needed

API: Application Programming Interface

RESTful API

- Representation State Transfer API
- A set of recommanded styles/rules for API design
    - Use of standard HTTP methods (GET, POST, PUT, DELTE, etc)
    - resources are idntified through URIs, where eacch URI represents a resources
    - statelessness, meaning the server does not store any client session information between requests
    - Use of standard HTTP status code to represent the request results

Example

1. Yes:

    • Retrieving User Information • Request: GET /users/123 • Description: Retrieves information about the user with ID 123 • Creating a New User • Request: POST /users • Request Body: json • { "name": "Alice", "email": "alice@example.com" } • Description: Creates a new user on the server.

    • Updating User Information • Request: PUT /users/123 • Request Body:json • { "email": "newemail@example.com" } • Description: Updates the email of the user with ID 123. • Deleting a User • Request: DELETE /users/123 • Description: Deletes the user with ID 123.

2. No:

    • Request: GET /getUser?id=123 • Description: Retrieves information about the user with ID 123. • Reason: Using Verbs in the URI • Request: POST /users/getUserInfo • Request Body: json • { "id": 123 } • Description: Retrieves information about the user with ID 123. • Reason: Using POST for Retrieval

    • Request: GET /users/updateEmail?id=123&email=newemail@example.com • Description: Updates the email of a user. • Reason: Including Actions in the URL