Group 10

Introduction to Computer Networks

Group Activity #02

Assignment Solutions

SWE3022 - Sungkyunkwan University

1. Network Application Architecture

A. Explain the client-server architecture. Suggest two example applications that use a client-server architecture.

Client-Server Architecture is a network paradigm in which services are provided by a central server and requested by multiple clients. The server is an always-on host with a permanent IP address, usually located in data centers for scalability and reliability. Clients, on the other hand, contact and communicate with the server to request resources or services. Unlike servers, clients may be intermittently connected to the network and often use dynamic IP addresses. Importantly, clients do not communicate directly with each other; instead, all communication goes through the server.

Examples are HTTP for web browsing and IMAP for Email services.

B. Explain the peer-to-peer (P2P) architecture. Suggest two example applications that use a P2P architecture.

Peer-to-Peer Architecture is a networking paradigm in which there is no always-on central server. Instead, arbitrary end systems, called peers, directly communicate with each other. In this model, peers both request services from other peers and provide services in return, which enables a decentralized exchange of resources. A key characteristic of P2P systems is self-scalability: when new peers join the network, they contribute not only additional demand but also new service capacity. However, peers are often intermittently connected and may change IP addresses, which makes management more complex compared to the client–server model.

Examples are P2P file sharing and BitTorrent.

2. Dynamic Adaptive Streaming over HTTP (DASH)

A. Explain DASH in server and client's side.

Server Side:

- Divides video files into multiple chunks (segments)
- Each chunk is stored and encoded at different bit rates/quality levels
- Creates a manifest file that provides URLs for different chunks and their encoding rates

A. Explain DASH in server and client’s side.

Client Side:

- Periodically measures server-to-client bandwidth
- Consults the manifest file to request one chunk at a time
- Chooses the maximum coding rate sustainable given current bandwidth conditions
- Can dynamically switch between different quality levels throughout playback
- Uses client-side buffering to ensure smooth playout

B. Specify the parts that ‘client’ determines in DASH.

A. When

- When to request each chunk (timing to prevent buffer starvation or overflow)
- When to switch quality levels based on bandwidth changes

B. What

- What encoding rate/quality level to request (higher quality when more bandwidth is available, lower quality when bandwidth is limited)
- What specific chunk to request next from the manifest

C. Where

- Where to request chunks from (can choose from different CDN servers/URLs)
- Which server location is "close" to the client or has high available bandwidth

C. While delivering the contents based on DASH, how does Content Delivery Network (CDN) help for efficient stream?

Content Distribution Networks fundamentally enable DASH streaming efficiency by solving the massive scalability challenge of distributing video content to users worldwide through strategically managed servers in multiple geographically distributed locations. CDNs store copies of video content across their distributed infrastructure and direct user requests to optimal locations for the best user experience.

This distributed approach employs one of two key philosophies:

- *enter deep*: Strategy pioneered by Akamai that places server clusters directly within access networks of Internet Service Providers across thousands of locations to minimize the number of links and routers between users and content, getting as close as possible to end users
- *bring home*: Approach used by Limelight and others that builds larger clusters at fewer sites, typically at Internet Exchange Points, trading some performance for reduced maintenance overhead

C. While delivering the contents based on DASH, how does Content Delivery Network (CDN) help for efficient stream?

The CDN infrastructure directly supports DASH's adaptive streaming by providing the geographic distribution that makes the client's "where to request chunks" decisions meaningful - when DASH clients measure bandwidth and request video chunks at appropriate quality levels, the CDN ensures these requests are served from nearby locations with optimal network paths. Additionally, CDNs employ intelligent content replication strategies, using pull-based approaches where clusters retrieve and cache popular content on-demand while removing less frequently requested videos, ensuring that the most relevant content is available close to users. This symbiotic relationship between DASH's client-side adaptation and CDN's distributed content placement enables the seamless delivery of millions of videos to hundreds of thousands of simultaneous users while maintaining the quality adaptation that makes modern video streaming possible.

3. Domain Name System (DNS)

A. Explain the Domain Name System (DNS).

Domain Name System(DNS) is a fundamental Internet service that translates human-readable names into IP addresses, which are used by hosts and routers to deliver data. Technically, DNS is a distributed database implemented in a hierarchy of many name servers. It operates as an application-layer protocol, where hosts and name servers communicate to resolve names into addresses (or vice versa). This allows users to use simple names while the underlying network uses numerical IP addresses for routing.

B. Explain the three classes of DNS servers, i.e., Root, TLD, and Authoritative DNS Servers.

Root DNS Servers are the highest level and direct queries to the correct Top-Level Domain(TLD) servers.

Top-Level Domain(TLD) Servers store imformation about authoritative servers for domains under their TLD and forward queries accordingly.

Finally, ***Authoritative DNS Servers*** hold the actual domain records and provide the final IP address, such as www.amazon.com.

C. Explain two DNS name resolutions such as iterative resolution and recursive resolution in detail.

There are two types of DNS name resolution: iterative and recursive.
In ***iterative resolution***, each contacted DNS server does not return the final answer but instead refers the client (or local DNS) to another server closer to the answer.(e.g., root → TLD → authoritatives).
In ***recursive resolution***, the queried DNS server takes full responsibility: it contacts other servers on behalf of the client and returns the final IP address. Iterative queries reduce load on upper servers, while recursive queries simplify the client's job but increase the burden on higher-level DNS servers.

C. Explain two DNS name resolutions such as iterative resolution and recursive resolution in detail.

There are two types of DNS name resolution: iterative and recursive.

In ***iterative resolution***, the client first asks its local DNS for the IP address of a domain. The local DNS then queries the root server, which does not give the final answer but instead points back to the local DNS with the address of the TLD server. The local DNS continues by querying the TLD server, which again refers it to the authoritative server for the domain. Finally, the local DNS queries the authoritative server and receives the final IP address, which it returns to the client.
(e.g., client → local DNS → root → local DNS → TLD → local DNS → authoritative → local DNS → client)

C. Explain two DNS name resolutions such as iterative resolution and recursive resolution in detail.

In ***recursive resolution***, the client asks the local DNS once for the IP address of a domain. This time, the local DNS takes full responsibility for the entire process. It queries the root server, then the TLD server, and finally the authoritative server, gathering the necessary information step by step. After receiving the final IP address, the local DNS responds directly to the client, so the client does not need to make any further queries.
(e.g., client → local DNS → root → TLD → authoritative → local DNS → client)

In addition, DNS uses ***caching*** to improve efficiency. Once resolved, a domain's IP can be reused from cache, avoiding repeated queries to root, TLD, and authoritative servers, thus reducing traffic and congestion.

4. Socket Programming

A. What does socket mean in computer network programming?

A socket is a door between an application process and the end-to-end transport protocol (TCP or UDP). It provides the interface through which applications send and receive data over the network.

B. Why do computer networks need sockets?

Standardized Interface: Sockets provide a common API so that applications can access network services without dealing with low-level protocol details.

Process Identification: Using the combination of an IP address and a port number, sockets uniquely identify processes across the network.

Transport Service Access: Sockets allow applications to utilize transport layer services such as reliable byte-stream transfer with TCP or fast datagram delivery with UDP

C. Explain the types of sockets on the Internet.

There are two main types of sockets used on the Internet: stream sockets and datagram sockets.

Stream sockets are based on TCP, which is connection-oriented. They provide reliable and in-order delivery of a continuous stream of bytes between two processes. This makes them suitable for applications like web browsing with HTTP or email transfer with SMTP.

Datagram sockets, on the other hand, are based on UDP, which is connectionless. They support message-oriented communication but do not guarantee reliability, ordering, or delivery. Because of their lightweight nature and low overhead, datagram sockets are often used in real-time applications such as video streaming and online games.

D. Make the description of how the following functions are used in C socket

- The **socket** function creates a new socket, which becomes the endpoint for communication. It returns non-negative socket descriptor on success, -1 on failure.
- The **connect** function is used by a client to initiate a connection to a server by specifying the server’s IP address and port number.
- The **bind** function assigns a local IP address and port number to a socket, usually on the server side, so that incoming requests can be directed properly.
- The **listen** function places the socket into a passive, listening state, which prepares the server to accept incoming connection requests.
- The **accept** function acknowledges an incoming client request and creates a new socket dedicated to communication with that client. The original listening socket remains available to wait for additional connections. It returns new non-negative socket descriptor on success, -1 on failure.