Application Layer

Madiba Hudson-Quansah

Contents

Chapter 1

Principles of Network Applications

Network applications are programs that run on end systems and rely on the network to provide services to users. Examples include web browsers, email clients, and file sharing applications. When writing network applications developers do not need to write software that runs on network-core devices, such as router or link-layer switches.

1.1 Network Application Architectures

Definition 1.1.1: Application Architecture

Designed by the application developer and dictates how the application is structured and how it communicates over the network.

There are two commonly used application architectures:

- Client-Server Architecture
- Peer-to-Peer Architecture

1.1.1 Client-Server Architecture

The server:

- Always-on host
- Permanent IP address, i.e. well-known address
- Often in data centres for scaling
- Serves requests from other hosts (clients)

The client:

- Communicates with server
- May be intermittently connected
- May have dynamic IP address
- Do not communicate directly with other clients

1.1.2 Peer-to-Peer Architecture

There is close to no reliance on dedicated servers in data centres, with clients directly communicating with each other as **peers**.

- Peers are not owned by the service provider

- Peers are intermittently connected with dynamic IP addresses
- Peers directly communicate with each other
- Self-scalable as more peers join the network and bring new service capacity as well as new services demands.
- Relatively cost effective as less server infrastructure is required.

1.2 Process Communicating

Definition 1.2.1: Process

A program running within a host. Processes on the same host can communicate with each other with interprocess communication (IPC). Processes on two different hosts communicate with each other by exchanging messages across the network.

Definition 1.2.2: Message

The data exchanged between processes.

Processes on different hosts communicate by exchanging messages across the computer network.

1.2.1 Client and Server Processes

Definition 1.2.3: Client Process

A process that initiates communication.

Definition 1.2.4: Server Process

A process that waits to be contacted.

Note:-

A process can be both a client and a server. For example, in P2P

Definition 1.2.5: Socket

An endpoint of a communication channel. Analogous to a door in the sense that a socket is where messages enter and leave the host. Sockets can be bound to a specific port number and an IP address, which is used to determine which process a message should be delivered to. Sockets are the interface between the application process and the transport layer protocol, such as TCP or UDP.

Definition 1.2.6: Port Number

A 16-bit number used to identify a socket. Port numbers are used to determine which process a message should be delivered to.

A network applications is made up of pairs of processes that send messages to each other over a network. Each process in a pair can be labelled as client or a server process, sometimes a process can be both a client and a server.

A process sends messages into and receives message from its **socket**. Each socket is identified by a **port number** that is used to determine which process a message should be delivered to.

In the internet hosts are identified by their IP addresses, which are 32-bit unique identifiers for hosts on the internet. Additionally to access a specific process on the host a **port number** is used. Thus, to access a process on a host the client needs to know the IP address of the host and the port number of the process. The combination of the IP address and port number is called a **socket address**.

1.3 Transport Services Available to Applications

A socket is the interface between the application process and the transport layer protocol, such as TCP or UDP. The services that a transport layer protocol can over to applications can be broadly classified along four dimensions:

- Reliable data transfer
- Throughput
- Timing
- Security

1.3.1 Reliable Data Transfer

Packets can get lost within a computer network, thus a transport layer protocol can provide reliable data transfer by implementing mechanisms to detect and recover from lost packets. This is know as guaranteed data delivery. When a transport protocol provides this service the sending process can just pass its data into the socket and known with certainty that the data will arrive without errors at the receiving process.

When a transport protocol does not provide reliable data transfer there is no guarantee that all the data sent by the client will arrive at the server. This may be acceptable for **loss-tolerant applications** like audio and video streaming, where the occasional loss of a packet may not be noticeable to the user. However, for applications that require reliable data transfer, such as file transfer or email, it is important to use a transport protocol that provides this service.

1.3.2 Throughput

Throughput is the amount of data that can be transferred from one process to another in a given amount of time. A transport protocol can provide a guaranteed available throughput at some specified rate. This is important for application that require a certain level of performance such as video streaming or online gaming. Such applications are called **bandwidth sensitive applications**. Applications that do not require a guaranteed level of performance are called **elastic applications**.

1.3.3 Timing

Timing guarantees refer to the ability of a transport protocol t provide guarantees on the maximum amount of time it takes for a message to be delivered from the sender to the receiver. This is important for applications that require real-time communication, such as video conferencing or online gaming. Such applications are called **delay-sensitive applications**

1.3.4 Security

A transport protocol can provide security services such as encryption and authentication to protect the confidentiality and integrity of the data being transmitted. This is important for applications that handle sensitive information, such as online banking or email. Such applications are called **security-sensitive applications**.

1.3.5 Transport Services Provided by the Internet

Application	Data Loss	Throughput	Time-Sensitive
File Transfer	No loss	Elastic	No
Email	No loss s	Elastic	No
Web Documents	No loss	Elastic	No
Video conferencing	Loss-tolerant	Bandwidth-sensitive	Yes
Streaming stored video	Loss-tolerant	Bandwidth-sensitive	No
Interactive games	Loss-tolerant	Bandwidth-sensitive	Yes
Online messaging	No loss	Elastic	Yes

1.3.6 TCP Services

The TCP service model is a connection-oriented service model, offering reliable data transfer.

Connection Oriented - TCP requires the client and server to exchange transport layer control information with each other before the exchange of messages. This is called a **handshake** an alerts the client and server allowing them to prepare for the data transfer. After the handshaking phase a TCP connection is established between the sockets of the two processes. The connection is a full-duplex connection, meaning that data can be sent in both directions at the same time. After the data transfer is complete the client and server exchange control information to terminate the connection.

Reliable Data Transfer - The communicating processes are guaranteed that all data sent by the client will arrive at the server without errors and in the correct order. TCP achieves this by implementing mechanisms to detect and recover from lost packets

Flow Control - TCP implements flow control to ensure that the sender does not overwhelm the receiver with too much data at once.

Congestion Control - TCP also implements congestion control to prevent the sender from overwhelming the network with too much data at once, which can lead to network congestion and packet loss.

Does not provide - Timing, guaranteed minimum throughput, security.

1.3.7 UDP Services

A lightweight transport protocol providing minimal services. UDP is connectionless, and unreliable meaning it does not guarantee the delivery of messages.

Connectionless - UDP does not require the client and server to exchange transport layer control information with each other before the exchange of messages. The client can just send a message to the server without establishing a connection first. This makes it faster than TCP for applications that do not require reliable data transfer.

Unreliable Data Transfer - UDP does not guarantee that all data sent by the client will arrive at the server without errors and in the correct order. This may be acceptable for loss-tolerant applications like audio and video streaming, where the occasional loss of a packet may not be noticeable to the user. However, for applications that require reliable data transfer, such as file transfer or email, it is important to use a transport protocol that provides this service.

Does not provide - Reliability, flow control, congestion control, timing, guaranteed minimum throughput, security, connection setup

1.3.8 Securing TCP

Default TCP and UDP sockets provide no encryption, meaning that data sent over a TCP connection is sent in clear text. This means that if an attacker is able to intercept the data being sent over the TCP connection, they can read the data, including any sensitive information such as passwords or credit card numbers. To address this issue, Transport Layer Security (TLS) can be used to provide encryption for TCP connections. TLS is a cryptographic protocol that provides data integrity, end-point authentication, and encryption for TCP connections. When TLS is used, the data sent over the TCP connection is encrypted, making it much more difficult for an attacker to intercept and read the data.

1.4 Application Layer Protocol

An application layer protocol defines how an application's processes, running on different end systems, pass messages to each other. It defines:

- The types of messages exchanged, i.e. request and response messages
- The syntax of the various message types, i.e. the fields in each message and how the fields are delineated
- The semantics of the fields, i.e. the meaning of the information in the fields and how the receiving process should interpret the information

- The rules for determining when and how a process sends and responds to messages, i.e. the order of message exchange and the actions taken when a message is received

The main difference between network applications and application-layer protocols is that an application-layer protocol is only a section of a network application. For example, a web browser is a network application that implements the HTTP application-layer protocol to communicate with web servers. The web browser also implements other protocols such as DNS to resolve domain names to IP addresses, and TLS to secure the communication with the web server.

There are two types of protocols

Open

Proprietary

Chapter 2

Web and HTTP

2.1 HyperText Markup Protocol (HTTP)

Definition 2.1.1: Object

A file, such as an HTML file, a JPEG image, a Java applet, or a video clip, that can be sent over the internet.

HTTP is implemented in two programs, a client program and a server program usually executing on different hosts. HTTP defines the structure of messages exchanged between the client and server and how the client and server should respond to various messages. HTTP is the foundation of data communication for the World Wide Web, and it is used to transfer web pages and other web resources between web servers and web browsers.

A web page consists of objects, which are files that can be sent over the internet. Examples of objects include HTML files, JPEG images, Java applets, and video clips. When a user requests a web page, the web browser sends an HTTP request to the web server, which then responds with the requested objects. The web browser then renders the web page using the received objects.

HTTP is based on TCP, i.e.:

- Client initiates TCP, creates socket, to a server at port 80
- Server accepts TCP connections from client
- HTTP messages are exchanged between browser and web server
- TCP connection is closed

TCP provides a reliable data transfer service to HTTP, which means that each request sent by a client eventually arrives intact at the server, and each response sent by the server eventually arrives intact at the client.

HTTP is stateless as the server maintains no information about past client requests. Each HTTP request is treated independently of any previous requests. This means that the server does not keep track of any information about the client or the client's previous requests.

2.1.1 Non-Persistent vs Persistent HTTP

In many applications the client and server communicate for an extended period of time, with the client making a series of requests and the server responding to each of the requests. Depending on the application the requests may be made back to back periodically at regular intervals or intermittently. This introduces the question of whether to use a single TCP connection for all the requests and responses, or to use a separate TCP connection for each request and response. These types of connections are known as persistent and non-persistent connections respectively. HTTP can be implemented using either persistent or non-persistent connections.

2.1.1.1 HTTP with Non-Persistent connections

In the case of a non-persistent connection a separate TCP connection is used for each request and response. This means that for each request the client establishes a new TCP connection to the server, sends the request, receives the response, and then closes the TCP connection.

HTTP/1.0 uses non persistent connections, supporting exactly one request and response per TCP connection.

Definition 2.1.2: Round Trip Time (RTT)

The time it takes for a small packet to travel from the client to the server and back again.

The RTT includes packet-propagation, packet-queuing, and packet-processing delays. Since TCP uses a connection-oriented service model, the client and server initiate a **three-way handshake** to establish a TCP connection before the client can send its request. The first handshake message (SYN) is sent from the client to the server, then the second handshake message (SYN-ACK) is sent from the server to the client, and finally the third handshake message (ACK) is sent from the client to the server with the request. This means at minimum two RTTs are needed before the client receives the response from the server.

The first RTT is for the three-way handshake to establish the TCP connection, and the second RTT is for the client to send the request and receive the response from the server. In the case of non-persistent HTTP, the RTT is incurred for each request and response, which can lead to significant delays if the client needs to make multiple requests to the server.

Non-Persistent HTTP is inefficient as it requires multiple TCP connections to be established and closed for each request and response, which can lead to significant delays due to the overhead of establishing and closing TCP connections, as well as the RTT for each request and response.

Non-Persistent HTTP connections also require TCP buffers and variables to be allocated for each connection, which can lead to increased memory usage on the server.

2.1.1.2 HTTP with Persistent connections

With HTTP/1.1, persistent connections are the default, meaning the server leaves the TCP connection open after sending a response. Subsequent requests and responses are sent over the same connection, avoiding the need for a new three-way handshake for every object. While the first object still requires 2 RTTs (handshake plus request), subsequent objects only require 1 RTT (or less with pipelining). Pipelining further improves performance by allowing the client to send multiple requests back-to-back without waiting for intervening responses.

2.1.2 Message Format

There are two types of HTTP messages: request messages and response

2.1.2.1 HTTP Request Message

Below is a typical HTTP request message:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

The components of a HTTP are:

Request Line - The first line of an HTTP request message. Contains three fields

- **Method** - The method field specifies the type of request being made. Common methods include GET, POST, and HEAD.
- **URL** - The URL field specifies the resource being requested. It includes the protocol (e.g. http), the hostname (e.g. www.someschool.edu), and the path to the resource (e.g. /somedir/page.html).
- **HTTP Version** - The HTTP version field specifies the version of HTTP being used by the client.

Header Fields - The subsequent lines after the request line. Each header field consists of a name and a value, separated by a colon. These fields provide additional information about the request, such as the client's capabilities and preferences.

Message Body - The message body is an optional part of the HTTP request message. It is used to send data from the client to the server, such as form data or file uploads. The message body is typically used with the POST method, but it can also be used with other methods as well.

2.1.2.2 HTTP Response Message

Below is a typical HTTP response message:

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

<html>
<body>
<h1>Welcome to my homepage</h1>
<p>My name is Madiba</p>
</body>
</html>
```

The components of a HTTP response message are:

Status Line - The first line of an HTTP response message. Contains three fields

HTTP Version - The HTTP version field specifies the version of HTTP being used by the server.

Status Code - The status code field is a three-digit code that indicates the status of the response. Common status codes include 200 (OK), 404 (Not Found), and 500 (Internal Server Error).

Reason Phrase - The reason phrase field is a brief description of the status code. It provides additional information about the status of the response.

Header Fields - The subsequent lines after the status line. Each header field consists of a name and a value, separated by a colon. These fields provide additional information about the response, such as the server's capabilities and preferences.

Message Body - The message body is an optional part of the HTTP response message. It is used to send data from the server to the client, such as the requested resource or an error message. The message body is typically used with successful responses (e.g. 200 OK), but it can also be used with error responses as well.

2.1.3 Cookies and State

Definition 2.1.3: Cookie

A small piece of data stored on the client's computer by the web browser. Cookies are used to maintain state and store information about the client, such as login credentials or user preferences.

HTTP is stateless, meaning that the server does not maintain any information across requests from the same client. Sometimes it is desirable for the server to maintain state and this is achieved using cookies.

Cookies are comprised of four components:

Cookie Header in Response - The Set-Cookie, header is a field in the HTTP response message that is sent from the server to the client. It contains the name and value of the cookie, as well as any additional attributes such as the expiration date or the domain for which the cookie is valid.

Cookie Header in Request - The Cookie, header is a field in the HTTP request message that is sent from the client to the server. It contains the name and value of the cookie that was previously set by the server. This allows the server to identify the client and maintain state across multiple requests from the same client.

Cookie File - The cookie file is a file stored on the client's computer by the web browser. It contains the name and value of the cookies that have been set by the server, as well as any additional attributes such as the expiration date or the domain for which the cookie is valid. The web browser uses the cookie file to manage cookies and send them back to the server in subsequent requests.

Backend Database - The backend database is a database maintained by the server that stores information about the clients, such as login credentials or user preferences. When a client sends a request to the server with a cookie header, the server can use the value of the cookie to look up the corresponding information in the backend database and maintain state across multiple requests from the same client.

2.1.4 Web Caching

Definition 2.1.4: Web Cache / Proxy Server

A network entity that satisfies HTTP requests on behalf of an origin web server.

When a browser is configured to use a web cache it goes through the following steps:

1. The browser establishes a TCP connection to the web cache and sends a HTTP request for the object to the web cache.
2. The web cache checks to so if it has a copy of the object stored locally. If it does the we cache returns the object within an HTTP response message to the client browser.
3. If the web cache does not have the object the web cache opens a TCP connection to the origin server and requests the object. After receiving the request the origin server services the request and sends the HTTP response to the web cache.
4. When the web cache receives the response it stores a copy in its local storage and sends a copy, within a HTTP response message, to the client browser.

2.1.4.1 Conditional GET

2.1.5 HTTP/2

2.1.5.1 HTTP/2 Framing

2.1.5.2 Response Message Prioritization and Server Pushing

2.1.6 HTTP/3

Chapter 3

Electronic Mail in the Internet

3.1 Simple Mail Transfer Protocol (SMTP)

3.2 Mail Message Formats

3.3 Mail Access Protocols

Chapter 4

Domain Name System (DNS)

4.1 DNS Services

4.2 DNS Architecture

4.2.1 Distributed Hierarchical Database

4.2.2 DNS Caching

4.3 DNS Records and Messages

4.3.1 DNS Messages

4.3.2 Inserting Records into the DNS Database

Chapter 5

Peer-to-Peer (P2P) File Sharing

5.1 Scalability

5.2 BitTorrent

Chapter 6

Video Streaming and Content Distribution Networks (CDNs)