Computer Networks

Sungchan Yi

February 2020

Contents

1 Introduction

1.1 What is the Internet?

1.2 Network Edge

1.3 Network Core

1.4 Internet Structure

1.5 Performance Metrics

1.5.1 Evaluation Metrics

- **Delay**: Packet delivering time from source to destination
- **Packet loss**: Ratio of lost packets to total sent packets[1]
  - If the queue is full, the arriving packets will be dropped
  - Lost packets may be re-transmitted by previous nodes, by source end system, or not at all
- **Throughput**: Amount of traffic delivered / unit time
  - Rate at which bits are transferred between source and destination
  - Can be measured instantaneously, or on average
  - *Bottleneck link*: The link on end-end path that constrains the throughput (usually the one with minimum capacity)

1.5.2 Four sources of delay

$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

- Queueing Delay
  - Time waiting at output buffer for transmission
  - **Congestion** dependent
- Transmission Delay
  - $d_{trans} = L/R$ where L: packet length (bits), R: link bandwidth (bps)
- Processing Delay
  - Bit error checking
  - Decision of output link
  - Typically takes less than a few milliseconds (hardware acceleration)
- Propagation Delay

[1] PDR: Packet delivery ratio

- $d_{prop} = d/s$ where d: length of physical link, s: signal speed

Queueing and transmission delay take up most of the delay.

1.5.3 More on Queueing Delay

$$\text{(Traffic Intensity)} = \frac{La}{R}$$

where R is the link bandwidth (bps) or transmission rate, L is the average packet length (bits), a is the average packet arrival rate. As the traffic intensity $\to 1$, the average queueing delay will grow without bound.

1.6 Protocol Stack

A communication protocol stack is composed of several **layers**. Each layer implements a service via its own internal actions, and by relying on services provided by the underlying layers.
Layering or **modularization** eases development, maintenance, and updating of the whole system. But this can be harmful in cases when a higher level layer needs information from the lower layers.[2]

1.6.1 Internet Protocol Stack

1. **Physical**: Bits on the wire
2. **Link**: Data transfer between neighboring network elements
3. **Network**: Routing of datagrams from source to destination
4. **Transport**: Process to process data transfer
5. **Session**: Synchronization, connection management, recovery of data exchange
6. **Presentation**: Allows applications to interpret the meaning of data
7. **Application**: Supporting network applications

1.7 Network Security

The field of network security arises from these questions:

- How can bad guys attack our computer networks?
- How do we defend our networks against those attacks?
- How do we design architectures that are immune to attacks?

[2] Consider a navigation system, which uses "physical" information like actual traffic, when choosing the fastest route between two places. But this "physical" information wouldn't normally be visible to other layers.

1.7.1 Forms of Attacks

- Malware
  - virus: A self-replicating infection by receiving or executing an object
  - worm: A self-replicating infection by passively receiving object that gets itself executed
  - spyware
  - ransomware
- Packet Sniffing: Promiscuous network interface reads/records all packets passing by
- Denial of Service: Attackers make resources unavailable by sending a huge amount of fake traffic
- Fake Addresses: Send packets with fake source address
- Fake Wi-Fi AP: Steal user's credentials using fake AP

1.8 History of the Internet

- Firstly developed as ARPAnet
- Internetworking architecture = autonomy + minimalism
- TCP/IP, WWW

2 Application Layer

Previously, we have seen that there are 5 layers in the Internet protocol. We will go through each layer, in a top-down approach. We will cover the application layer in this section.

2.1 Principles of Application

2.1.1 Network Applications

- Types: email, web (server software, browser), P2P file sharing, SNS, messenger program, online games, streaming stored video (YouTube, Netflix)
- Run on (different) **end systems**: network core devices *do not* run user applications. Ex) Routers only transfer information
- Communication over network (between end systems)

2.1.2 Application Architectures

There are two kinds of application structures: *client-server* model, and *peer-to-peer* model

- **Client-Server Model**
  - The **server** is *always on*, has permanent IP address, since clients must be able to access the server anytime

  - For scaling (to support a huge number of clients), a data center is typically built
  - The **client** is a program that communicates with the server. It may be intermittently connected, and may have dynamic IP addresses. For communication, the client must send a request to the server first, using its IP address. Then the server will respond to that IP address.
  - Clients do not communicate directly with each other
- **Peer-to-Peer Model** (P2P)
  - There is no *always on* server. Each clients can function both as a client and a server. Arbitrary end systems will directly communicate with each other.
  - Peers are intermittently connected and they can change IP addresses, so it is complex to manage.
  - **Self scalability**: New peers bring new service capacity, as well as new service demands

2.1.3 Application Layer Protocol

The application layer is on the top of the Internet protocol. Then, the applications on this layer will communicate with the application layer on another end device. The protocol defined for this communication is called the **application layer protocol**. There are two types of messages that network application protocols exchange - **request**s and **response**s.

- Message **Syntax**: What kinds of fields are there in the messages? How are the fields delineated?
- Message **Semantics**: What is the meaning of the information in the fields?
- Message **Pragmatics**: When and how do we process (send/respond) the messages?

2.1.4 Application Protocol

- Open Protocols
  - Standardized, open to public, for interoperability.[3]
  - Even if some non-authorized clients create a message that obeys the protocol, they can communicate with the server.
  - Ex) HTTP, SMTP
- Proprietary Protocols
  - A protocol specific for some program
  - Ex) Skype
  - We cannot communicate with Skype without using the Skype application

[3] **Interoperability** is a characteristic of a product or system, whose interfaces are completely understood, to work with other products or systems, at present or in the future, in either implementation or access, without any restrictions.

2.1.5 Requirements of Network Applications

Application	Data Loss	Throughput	Time Sensitive
File Transfer	×	elastic	×
Email	×	elastic	×
Web Documents	×	elastic	×
Realtime Audio/Video	loss-tolerant	Audio: 5kbps∼1Mbps / Video: 10kbps∼5Mbps	100ms
Stored Audio/Video	loss-tolerant	(same)	A few secs
Interactive Games	loss-tolerant	$\geq$ Few kbps	100ms
Text Messaging	×	elastic	Yes and no

The applications that require correct communication of information have *elastic* throughput, since the correctness of the information is a lot more important than the speed of communication.

These requirements should be met by the *transport layer protocols*. The throughput and other requirements highly depend on the physical devices (routers, cables etc.) that hold up the network structure. The developers on the application layer cannot handle these requirements properly.

2.1.6 Transport Protocol Services

- **TCP Service** (Transmission Control Protocol)
  - **Error control**: In charge of reliable transport between sending and receiving processes
  - **Flow control**: The sender won't overwhelm the receiver (not too much data)
  - **Congestion control**: Throttle sender when network is overloaded
  - *Does not provide*: Timing, minimum throughput guarantee, security
  - *Connection oriented*: Setup is required between client and server processes
- **UDP Service** (User Datagram Protocol)
  - Unreliable data transfer between sending and receiving processes
  - Why do we use UDP? - Some programs may require UDP. For example, the error controlling in TCP lowers the throughput, but UDP will ignore this error, resulting in faster communication, which is suitable for multimedia programs

2.2 Web and HTTP

Web pages consist of **objects**. They can be an HTML file, JPEG image, Java applet, audio file, and more. Web page is described by **HTML-file**(s) which include several referenced objects. Each object is addressable by a **URL**(Uniform Resource Locator).

$$\overbrace{\texttt{www.someschool.edu}}^{\text{host name}}\,/\,\overbrace{\texttt{someDept/pic.gif}}^{\text{path name}}$$

2.2.1 HTTP Overview

- **HTTP** (HyperText Transfer Protocol)
  - Web's application layer protocol
  - **Hyperlink**: A reference to data the reader can directly follow by clicking
  - Uses client-server model
  - Client uses a browser that requests, receives, and displays the Web objects
  - The server is a web server that sends objects in response to request from clients
- Based on **TCP**
  1. Client initiates TCP connection (socket connection) to server
  2. Server accepts TCP connection from client
  3. HTTP messages are exchanged between brower and Web server
  4. TCP connection is closed
- *HTTP reponse time* (**RTT**: Round trip time)
  - 1 RTT to initiate TCP connection
  - 1 RTT for HTTP request and first few bytes of HTTP response to return
  - File transmission time
  - Total = 2 RTT + file transmission time

2.2.2 HTTP Version

- **HTTP/1** (1996)
  - *Non-persistent* HTTP: Single object per single TCP connection
  - Long latency
- **HTTP/1.1** (1999)
  - *Persistent* HTTP: Multiple objects over on TCP connection
  - Decreased latency
  - *Synchronous* order of response/request pairs over one TCP connection
- **HTTP/2** (2015)
  - *Persistent* HTTP
  - *Asynchronous* (parallel) multiple response/request pairs over one TCP connection

2.2.3 HTTP Message

There are two types of messages: **request**s and **response**s.

- **Request Message**: Request line + Header lines + Body
- **Response Message**: Response line + Header lines + Body
- Request line: method, URL, version
- Response line: version, status code, status text
- Header lines: header field name, value
- Body: entity body

2.2.4 REST

- **REpresentational State Transfer**
  - HTTP should be **stateless**. If the state of client is kept in the server, it causes overhead for the server.
  - The server will not store each client's states. Instead, the server will store the information about the client in the HTTP message. The server should also store the method to interpret the message. With all these information, the client knows how to fetch the data.
  - We call a service **RESTful** if the service conforms to this architectural style
- **Architectural Constraints**
  - *Client-server architecture*: Separation of the user interface concerns from the data storage concerns
  - *Statelessness*: No client context should be stored on the server between requests
  - *Cacheability*: Server responses are cachable on client and intermediaries
  - *Layered system*: One should be unable to tell whether a client is directly connected to the end server or to an intermediary along the way
  - *Uniform interface*: Simplification and decoupling of the architecture (Use of standardized languages - HTML, XML, JSON - that is not restricted by some computer architecture)
  - *Code on demand* (optional): Should be able to transfer executable code such as Java applets and JavaScript

2.3 Cookies and Web Caching

2.3.1 Cookies

Cookies keep the states of clients.

1. Client has a cookie file
2. A usual HTTP request message is sent to the server (for the first time)
3. The server creates an ID for the user, and creates an entry in the database

4. The server responds with the ID, and tells the client to set the cookie with the given ID
5. Now the client can send HTTP requests using that ID inside the cookie file
6. The server (database) performs a cookie-specific action (distinguished by ID), and responds as usual
7. A week later, (if the cookie still exists) the cookie can be used again for communication with the server

2.3.2 Web Caching

- For some servers (sites) with lots of visitors, request and responses for the exact same site would cause a huge overhead.
- The server prepares a **proxy server** that caches this information
  - If the requested object is not in the cache, the proxy server requests the object from the origin server, and caches the data (and also responds to the client)
  - Otherwise, the proxy server will used the cached data to respond to the client request
- Effects of Web Caching
  - For clients, the response time is reduced
  - For servers, it can handle more users (reduced request overheads)
  - For local ISPs, the traffic to external server is reduced, so the *access link* can be used efficiently

2.4 SSL/TLS

2.5 Electronic Mail

2.6 Domain Name System

2.7 Peer-to-Peer Application

2.8 Video Streaming and CDNs