Computer Networks

Sungchan Yi

February 2020

Contents

1 Introduction

1.1 What is the Internet?

- **Internet** = Inter- + net(work)
- Network of networks
- Various types of networks: mobile network, home network, global ISP, regional ISP

1.1.1 Components

- HW components
  - End hosts
  - Links: copper, fiber, radio, satellite
  - Interconnection devices: router, switch, repeater
- SW components
  - Operating software
  - Application programs
  - **Protocols**

1.1.2 Protocols

- A defined set of rules and regulations that determine how data is transmitted in telecommunications and computer networking
- All communication activity in Internet is governed by protocols
- Protocols define
  - Message format
  - Order of messages sent and received among network entities
  - Actions taken on message transmission and receipt

1.2 Network Edge

Network edge contain hosts, which are clients and servers.

1.3 Network Core

Network core is a mesh of interconnected routers or switches. They forward packets from one router (or switch) to the next along the path from the source to destination

1.3.1 Switching Mechanisms

- **Circuit Switching**
  - End to end resources reserved for communication between source and destination
  - Entire data flow through along the path like water
  - Resources are dedicated to each connection - the circuit segment is idle if it is note being used
  - Common in traditional telephone networks
  - Channel allocation methods: frequency division multiplexing (FDM), time division multiplexing (TDM)
- **Packet Switching**
  - Entire data is broken into small packets
  - Each packet has its destination address
  - Each packet is handled independently
  - Packet is transmitted at full link capacity
  - Takes L/R seconds to transmit L-bit packet into link at R bps
  - *Store and forward*: entire packet must arrive at the router before it can be forwarded to the next
  - End to end delay $\approx 2L/R$
  - If arrival rate of packets exceed the transmission rate of link, packets can be dropped (lost) if the packet queue inside the router is full
- Comparison
  - Packet switching allows more users to use the network
  - Circuit switching guarantees the quality of service for each connection

1.4 Internet Structure

Nobody or everybody is in charge of the Internet.

- End systems connect to the Internet via **access ISPs** (Internet Service Providers)
- Access ISPs in turn must be interconnected so that any two hosts can communicate with each other
- Resulting network of networks is very complex

1.4.1 Connecting Access ISPs

- How do we *interconnect* the access ISPs?
- Connecting each access ISP to every other one would not be scalable, since we need $\mathcal{O}(n^2)$ connections
- Better: Keep a *global transit ISP* and connect each access ISP to it
- Competing global transit ISPs appear and the are also interconnected by peering link and Internet exchange point (IXP)
- Regional networks arise to connect access networks to global ISPs

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
  - $d_{prop} = d/s$ 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 $\rightarrow$ 1, the average queueing delay will grow without bound.

[1] PDR: Packet delivery ratio

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?

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

[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.

1.8 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

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	≥ 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

[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.

  - *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

2.4.1 Securing TCP

- We often use TCP and UDP for transport layer protocols. But when these protocols were developed back then, security considerations were not taken into account
- TCP and UDP have no encrpytion, even passwords traversed the Internet in clear text
- **SSL/TLS** provides encrypted TCP connection at the *application layer*
- Assures data integrity[4], and end-point authentication
- **SSL** (Secured Socket Layer)
  - SSL v2.0 and v3.0 were released in 1995 and 1996, respectively
- **TLS** (Transport Layer Security)
  - Improved version of SSL v3.0
  - More secure than SSL, but slower due to the two step communication processes

2.4.2 SSL/TLS Principle

1. When client connects to a server, the client asks for a secure SSL session
2. The server sends a certificate[5], that contains the server's public key
3. The client sends a one time encryption key for the SSL session, encrypted with the server's public key
4. The server decrypts the session key using its private key and establishes a secure session
5. Now the client and the server can safely send/receive encrypted data

HTTPS is HTTP + SSL/TLS

2.5 Electronic Mail

2.5.1 Electronic Mail

- There are 3 components that consist electronic mail
  - *User agents* (clients): edit mail, read mail
  - *Mail servers*: Google, Daum, Naver etc.
  - *Protocols*: SMPT, POP3, IMAP
- Components of mail servers
  - A mailbox for incoming messages
  - A message queue for outgoing messages
  - A protocol for exchanging mail between mail servers

[4] Data doesn't change during transmission

[5] The client must check that this certificate is valid, and this certificate has to be signed by someone that the client trusts.

2.5.2 SMTP Protocol

- **SMTP** (Simple Mail Transfer Protocol) [RFC 2821] uses TCP as the transport layer protocol for reliable email delivery
- Three phases of transfer
  - Handshaking (Check sending server and receiving server)
  - Transfer of messages
  - Closure
- Command/Response interaction (like HTTP)
  - Commands are in ASCII text
  - Reponses contain status codes and phrase

2.5.3 Mail Access Protocol

- **POP3** (Post Office Protocol 3)
  - By default, deletes messages from the server after retrieving data
  - Disconnects from the server after download
  - Needs to reconnect to the server on each download
- **IMAP** (Internet Mail Access Protocol)
  - Keeps all messages at the server and allows user to organize message folders
  - Support synchronization across multiple devices
  - Stays connected until the mail client app is closed and downloads messages on demand
- **HTTP**
  - Web-based email
  - Used between browser and server
  - Hotmail in the mid 1990s
  - Google, Yahoo, etc.

2.6 Domain Name System

To connect on the Internet, the client must know the *address* of the server. This *address* is called an **IP address**, and it is represented by 4 numbers between 0 and 255. But since these 4 numbers are hard to remember, (imagine having to memorize addresses for each website you use!) people use the *name* of servers, instead of IP addresses.

2.6.1 Domain Name System

- DNS **translates** an hostname to an IP address
- Procedure (Simplified)
    1. The client asks the DNS for the IP address of some website
    2. The DNS responds with the IP address of that website
    3. The client uses that IP address to connect to the website
- Achieves *load distribution* - many IP addresses correspond to one hostname (replicated Web servers), thus requests to the same hostname can be distributed between multiple servers
- **Distributed database system**
- De-centralized system
    - Not scalable (for large number of clients)
    - Single point of failure: The whole system breaks down when the centralized system fails
    - Traffic volume: Bottleneck
    - Distant centralized database: Longer delay for connections from far places from the system
- Therefore uses **distributed** & **hierarchical** database
    - Root DNS server on the top
    - Top-level domains (TLD) on the next level (`com`, `org`, `edu`)
    - Authoritative domains on the next level
- Database Query Procedure (Simplified)
    1. Client wants IP for `www.amazon.com`
    2. Client queries root DNS server to find `com` DNS server
    3. Client queries `com` DNS server to get `amazon.com` DNS server
    4. Client queries `amazon.com` DNS server to get the IP address for `www.amazon.com`

2.6.2 DNS Hierarchy

- **Root Name Servers**
    - Directly answers requests for records in the root zone
    - Answers requests by returning a list of the authoritative name servers for the appropriate top-level domain
    - 13 of them worldwide
- **Top Level Domains** (TLD)
    - Responsible for `com`, `org`, `net`, `edu` ...

  - All top-level contry domains like `uk`, `fr`, `ca`, `kr` ...

- **Authoritative Servers**
  - An organization's own DNS server(s), prociding authoritative hostname to IP mappings for organization's named hosts
  - Maintained by organization itself or service provider

- Local DNS Servers
  - Does not strictly belong to DNS hierarchy
  - Each ISP has on also called "default name server"
  - When the client makes a DNS query, it is sent to its local DNS server
  - The local DNS server caches the results of recent queries (name to address translation pairs)
  - Can also act as a proxy, forwards query into hierarchy

2.6.3 DNS Name Resolution

- Situation: Host at `cis.poly.edu` wants IP for `gaia.cs.umass.edu`

- **Iterated Query**
  - Contacted server replies with the name of server to contact
  - *I don't know this name, but ask this server*
  - In the procedure below, we assume that no results are cached. If there is a cached result during the process, that result can be used. Then some steps can be skipped

  1. Client sends a request to local DNS server
  2. The local DNS server asks the root DNS server for the `edu` TLD DNS server
  3. The root DNS server replies
  4. The local DNS server asks the `edu` TLD DNS server for the authoritative DNS server
  5. The local DNS server asks the authoritative DNS server for the IP of `gaia.cs.umass.edu`
  6. The local DNS server replies to the client with the IP address

- **Recursive Query**
  - Contacted server requests another server
  - Puts the burden of name resolution on the contacted name server
  - Not recommended due to heavy load at upper levels of hierarchy and security issues
  - Systems on upper levels must wait for the result of each query - may result in a DoS attack

  1. Client sends a request to local DNS server
  2. The local DNS server asks the root DNS server

3. The root DNS server asks the `edu` TLD DNS server
4. The `edu` TLD DNS server asks the authoritative DNS server
5. The authoritative DNS server replies with the IP address
6. The `edu` TLD DNS server responds to the root DNS server with the IP address
7. The root DNS server responds to the local DNS server with the IP address
8. The local DNS server replies to the client with the IP address

2.6.4 Attacking DNS

- DDoS Attacks
  - Attacking root DNS servers with traffic - doesn't work well since local DNS servers cache IPs of TLD servers (doesn't connect to the root server)
  - Attacking TLD servers can be more dangerous
- Amplification Attacks
  - Tricks the DNS server by putting the victim's IP inside the query
  - The DNS server will send the result of the query to the victim's IP
- Pharming Attacks
  - Private data + Farming
  - Domain hijacking, DNS poisoning
  - The attacker breaks into the local DNS and modifies some mapping to the attacker's fake website
  - The user may connect to that fake website and may enter private information

2.7 Peer-to-Peer Application

2.7.1 P2P Architecture

- No always-on server
- Arbitrary end systems communicate directly
- Peers are intermittently connected and change IP addresses
- P2P is scalable, compared to client-server model

2.7.2 BitTorrent

- File is divided into 256 KB chunks
- Peers in torrent send/receive file chunks
- Torrent: group of peers exchanging chunks of a file

- Tracker: tracks peers participating in torrent
- A new client arrives, obtains list of peers from tracker, and begins exchanging file chunks with peers
- Chunk Receiving
  - At any given time, different peers have different subsets of file chunks
  - Periodically, client asks each peer for list of chunks that they have
  - Client requests missing chunks from peers, *rarest first*
  - Then the rarity of chunks will decrease, and be more available to other peers
- Chunk Sending: *tit-for-tat*
  - To prevent free-riders (people who download but do not provide content)
  - The sender sends chunks to 4 other peers that are currently sending chunks to itself at the highest rate
  - Other peers will not receive chunks from this sender
  - The top 4 peers are evaluated every 10 seconds
  - For newly connected peers to receive chunks, the sender selects a random peer every 30 seconds, and starts sending chunks to it
  - Now the newly connected peer may be included in the top 4 peers
  - More upload results in better peers, resulting in faster download of data

2.8 Video Streaming and CDNs

2.8.1 Content Distribution Networks

- Video traffic is the major consumer of Internet bandwidth
- Challenge: **Scalability** - If we only use a single video server ...
  - Single point of failure
  - Point of network congestion
  - Longer path to distant clients
  - Multiple copies of same video sent over outgoing link
- Solution: *multiple copies of videos at multiple geographically distributed sites*
- CDN servers contain multiple copies of the same content, and lets the client stream the content data from the nearest/fastest CDN server

3 Transport Layer

3.1 Transport Layer Services

Some terminology:

- **Program**: An executable file containing a set of instructions written to perform a specific job (usually stored on a disk)
- **Process**: An executing *instance* of a program that resides on the primary memory. Several processes can be related to the same program at the same time
- **Thread**[6]: The smallest executable unit of a process

3.1.1 Transport Layer Function

- Provides **logical communication between processes**
- The layer relies on and enhances services from the network layer[7]
- Sending Side
  - Applies **fragmentation** to application messages
  - Passes segments[8] to network layer
- Receiving Side
  - **Reassembles** segments into messages
  - Passes the assembled message to the application layer

3.1.2 TCP and UDP

- **Transmission Control Procotol** (TCP)[9]
  - **Reliable**, **in-order**[10] delivery
  - **Connection oriented** service: connection setup, error control, flow control, congestion control
- **User Datagram Protocol** (UDP)
  - Unreliable, unordered delivery
  - Connection-less service: faster than TCP

[6] **Thread** of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system.

[7] The network layer provides logical communication between **hosts**

[8] We see terms like *frame*, *datagram*, *packet*, *segment* when we study computer networks. They are similar, but we use layer-specific terms to represent the information unit in that layer. In the transport layer, we use the term **segment**.

[9] Refer to 2.1.6.

[10] This doesn't mean that the order of sent messages is always preserved when receiving. (physically) In the application layer's point of view, it just *seems* like it's received in the same order.

3.2 Multiplexing and Socket

3.2.1 Multiplexing and Demultiplexing

- This is the most basic role of the transport layer
- **Multiplexing** at sender: the sender sends data from its own multiple applications through network
- Data from multiple services are sent through a single shared channel
- **Demultiplexing** at receiver: the receiver delivers data packets to their appropriate receivers among its own multiple applications
- **Port numbers** are used for demultiplexing
  - Different applications are assigned to different port numbers
  - Transport layer segments have fields for source/destination port numbers in common
  - Used to differentiate segments
- Note that for connection oriented protocols (like TCP), the source IP and port are also used to differentiate each connection[11]
- But how does the sender know the destination port on the receiver?

3.2.2 Sockets

- API between application layer and transport layer
- Processes send/receive messages to/from its **socket**
- Analogous to door
  - Sender passes message through the door
  - Sender relies on transport infrastructure on other side of the door to deliver message to socket at the receiving process
- The *socket* is provided as the form of APIs by the operating system

3.3 User Datagram Protocol

- Only provides the basic functions (multiplexing)
- **Connection-less** service
  - Each UDP segment is handled independently of others
  - *Unreliable*: UDP segments may be lost or delivered out of order to app
- But since UDP is fast, it is used for streaming multimedia applications, DNS, and SNMP

[11] Multiple applications can listen on the same port.

- For reliable transfer over UDP, the application must add that function (such as application specific error recovery)
- **UDP segment header**: Source port (16 bits), Destination port (16 bits), length, checksum, payload
- **Advantages**
  - No connection establishment (no delay)
  - Simple: No connection state at sender/receiver
  - Small header size[12]
  - No congestion control: UDP can blast away as fast as desired
- **UDP Checksum**
  - Detects transmission errors
  - UDP doesn't have to provide reliable connections, but this checksum can be used to provide additional features elsewhere
  - Sender creates a 16 bit integer checksum code of segment contents including the header
  - The receiver will compute the checksum of the received message, and checks if the computed value is equal to the received value

3.3.1 Checksum Method

- Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.[13]

3.4 Reliable Data Transfer Principles

3.4.1 Principles of Reliable Data Transfer

- Service abstraction (provided to the upper-layer) through a reliable channel
- *Service model of TCP*: No corruption and no loss of data, delivered in the order in which they were sent
- The lower layers of the network (below transport layer) is unreliable, but the TCP protocol in the transport layer will pre-process any existing errors and pass them onto the receiver.

3.4.2 Error Types and Solution

- **Bit Error**
  - Some of the bits are changed
  - This can be checked by comparing the checksum in every segment
  - If the receiver successfully received the packet, **ACK**(acknowledgment) message is sent to the sender

[12] Compare this to TCP headers.

[13] RFC 768

- **Packet Loss** (Data of ACK)
  - Packet is gone, the receiver doesn't receive the packet
  - *Timeout* of sender's timer - the sender sends the packet and waits for ACK, but if ACK doesn't arrive, the sender will re-send the packet
  - But consider the case where the returning ACK is lost - the sender will re-send the packet anyways, but how does the receiver know that this packet is a duplicate or not?
  - To solve this problem, *packet sequence number* is used
  - Suppose the receiver received packet k, and sent an ACK message. If packet k arrives again, the receiver will know that this packet was re-sent
  - Also, for transmitting large data, the data will be segmented and labeled with a packet sequence number. Then when the receiver receives the data, it will be possible to re-assemble the original data
  - Packet sequence numbers allow *ordered delivery* and data duplication prevention

3.4.3 Automatic Repeat Request

- For communication error recovery, we need a packet retransmission method
- We use **ARQ**(Automatic Repeat reQuest)
- **Stop and wait**: sending and checking one segment at a time
- **Pipelining method**: sending and checking multiple segments at a time (*go back N*, *selective repeat* method)

Stop and Wait

- Sender sends a packet and waits for the receiver's response with ACK
- After receiving the ACK message, the sender will send the next packet
- If the sender doesn't receive the ACK message, the sender will re-send the previous packet
- The receiver responds with ACK if the calculated checksum matches the checksum value in the segment
- The length of *timeout* t is the main problem!
- If t is too long, the sender has to *wait* for that amount of time which will slow down the transmission.
- If t is too short, a normal transmission may be thought of as a timeout (premature timeout). This may happen when the network is too busy. The ACK message couldn't arrive on time. This will cause the sender to re-send the same message again (often), which is a waste of network resource
- Thus when setting the timeout time, the *round trip time* between the sender and the receiver should be considered
- Performance Analysis
  - 1 Gbps link, 15 ms propagation delay, 1 kB packet

$$d_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits}/s} = 8\mu s$$

  - **Utilization**: fraction of time sender busy sending

$$U = \frac{d_{trans}}{\text{RTT} + d_{trans}} = 0.00027$$

  - If RTT[14] is 30 ms, 1 kB packet is sent every 30 ms, *so 33 kb/s throughput over 1 Gbps link*[15]
  - Usually d_{trans} is very small (fast link), compared to RTT. Thus $U \approx 0$, which means that the time spent in sending the packet is nearly 0%
  - Note that the size of ACK message is very small, thus it is ignored here

Pipelined Protocols

- **Pipelining**: multiple packets can be sent and received
- Range of sequence numbers must be increased
- Buffering is necessary at sender and receiver
- Suppose we send n packets in the above example. (n-packet pipelining) Then the utilization[16] would be

$$U = \frac{n \cdot d_{trans}}{\text{RTT} + d_{trans}} = 0.00027n$$

  which is a lot better. Now we try to increase the value of n

Go Back N

- The sender can have up to N unacknowledged packets in pipeline at once
- The receiver only sends **cumulative ACK** - if the receiver finds an error in packet k, then the sender has to re-send packets $k, \dots, n$
- The cumulative ACK will mean: "transmit successful, up to these packets"
- The sender has a timer for *oldest* unacknowledged packet, and when timeout occurs, the sender will re-send all unacknowledged packets

Selective Repeat

- The sender can have up to N unacknowledged packets in pipeline at once
- The receiver sends **individual ACK** for each packet - if the receiver finds an error in packet k, only that packet needs to be re-sent
- The sender has a timer for each unacknowledged packet, and when timeout for packet k occurs, only packet k needs to be re-sent

[14] Round trip time

[15] What a waste of resources!

[16] The denominator doesn't change because we measure the time from [the moment that the first packet was sent], to [the time when ACK for the first packet arrived].

3.4.4 Go Back N - In Detail

Sender's Perspective

- Packet sequence number is contained in each packet header
- A **window** is defined - the number of consecutive unacknowledged packets allowed

1. Let the window size be n, and suppose we have sent up to k ($k \leq n$) packets.
2. The receiver will receive the data, and send a cumulative ACK for packet i ($i \leq k$)
3. Since the cumulative ACK for packet i has arrived, packets $1, \ldots, i$ are acknowledged
4. The timer will be moved to packet $i + 1$ and wait for the next ACK. If the ACK does not arrive in time, packets $i + 1, \ldots, k$ will be re-sent

- Note that k (number of sent packets) can keep changing in the process above, just keep in mind that the acknowledged packet number i should be less than k
- Furthermore, the window size n is not fixed. Hence the term **sliding window**
- The larger the window size, higher the throughput, but the number of packets to re-send on an error will also increase
- The window size should be controlled so that it does not cause network congestion or receiver buffer overflow

Receiver's Perspective

- When packet k arrives,
  - *If*: Packet $k + 1$ arrives shortly[17] after packet k has arrived, wait for the next packet
  - *Else if*: Packet $k + 1$ doesn't arrive (due to congestion or other reasons), respond with *cumulative* ACK for packet k and wait for the next packet
  - *Else*: A packet other than $k+1$ arrives (ex. packet $k+1$ has been lost), discard the packet[18] and re-send the last sent ACK
- The *Else* case is the only problem. The sender would be expecting ACK for packet $k + 1$, but the sender will get the ACK for the last successful transmission[19]
- Then packet $k + 1$ will cause timeout and the sender will resend the packets from $k + 1, \ldots$

[17] Depends on the receiver's settings

[18] It's going to be re-sent from the sender anyways

[19] This duplicate ACK will be ignored

3.4.5 Selective Repeat - In Detail

- The receiver will buffer the packets, in case some packet is lost - the receiver waits for that packet to be re-sent, and when the receiver gets it, it can pass that information to the application layer[20]
- **Maximum packet sequence number $\geq$ 2 $\times$ window size**
- For example, with 4 sequence numbers 0, 1, 2, 3 and window size 3, there could be a case where
  1. The sender sent packets 0, 1, 2
  2. *The receiver got those packets and now expects packets 3, 0, 1*
  3. Unfortunately, the ACK for packets 0, 1, 2, do not arrive to the sender
  4. The sender re-sends packets 0, 1, 2 - which are the same packets from step 1
  5. The receiver has no idea that this packet is a re-sent version!
  6. The receiver will accept the re-sent packet 0 as the next packet,
- The receiver doesn't know what's happening on the senders side. It can only distinguish the packets by the packet sequence number. If the packet sequence number is too small for the window size, data may be corrupted during transmission

3.5 Transmission Control Protocol

3.6 Congestion Control

3.7 TCP Congestion Control Algorithm

3.8 TCP vs UDP

[20] Recall that transport layers had to re-assemble the packets before passing them to the application layer, so that it would look like the packets were received in order